

MTT39/CN
Rev. 3.4

TPU

MICROCODE

TABLE OF CONTENTS

including directory of main graphics

- **Chapter 1: Host Interface**

 - Emulation Capability

 - TPU Simplified Block Diagram

 - TCR1 & TCR2 Clocking

 - TPU Register Map

 - TPU Channel Map

- **Chapter 2: TPU Microcode Overview**

 - Instruction Format-general form

 - Specific Instruction Formats

- **Chapter 3: Entry Points**

 - Entry Table chart

 - Entry Point general form & examples

- **Chapter 4: RAM**

 - Parameter RAM map

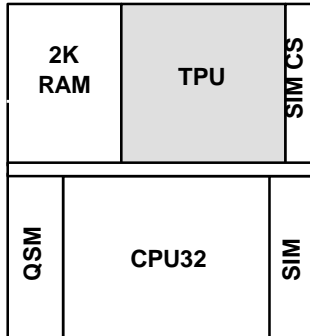
TABLE OF CONTENTS continued

- **Chapter 5: Channel Control**
 - Channel Hardware
 - Transition Event Summary
 - Match Event Summary
- **Chapter 6: Execution Unit**
 - Register List
 - Execution Unit Hardware
 - CHAN_REG Chart (changing environ.)
- **Chapter 7: Microengine/Sequencing**
 - Branch Chart
- **Chapter 8: Scheduler**
 - Priority Scheme
 - Worst Case Latency
- **TPU Quick Reference Guide**

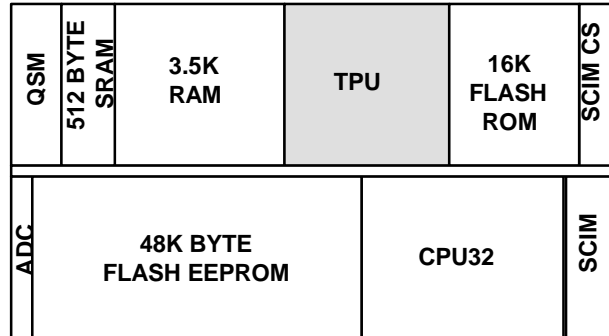
HOST INTERFACE

Time Processor Unit

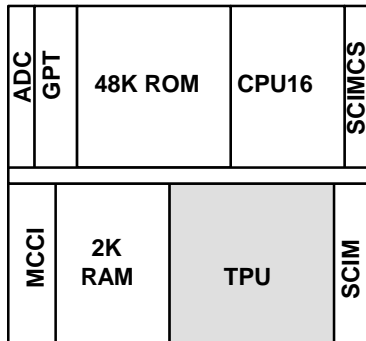
68332



68F333 ORION



68HC16Y1 TUNA



Overview

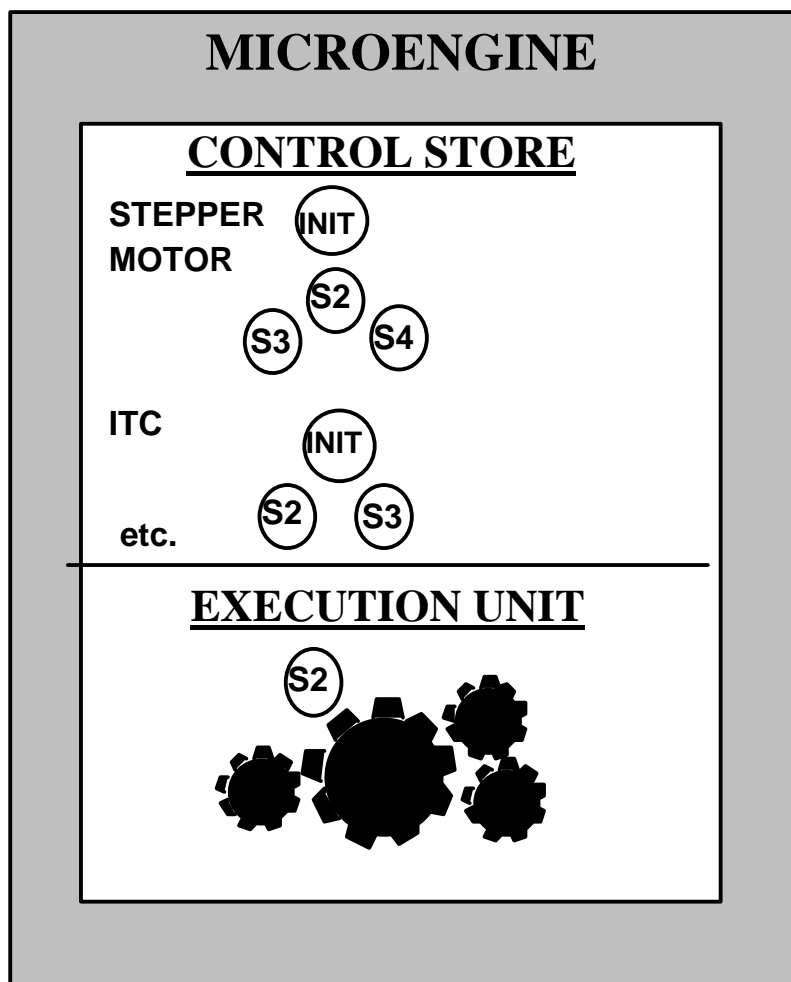
TPU : performs **time functions**.

Time function = states.

State = microinstructions.

Execution unit services a time function one state at a time.

Within TPU:



TPU Time Functions

Standard Mask:

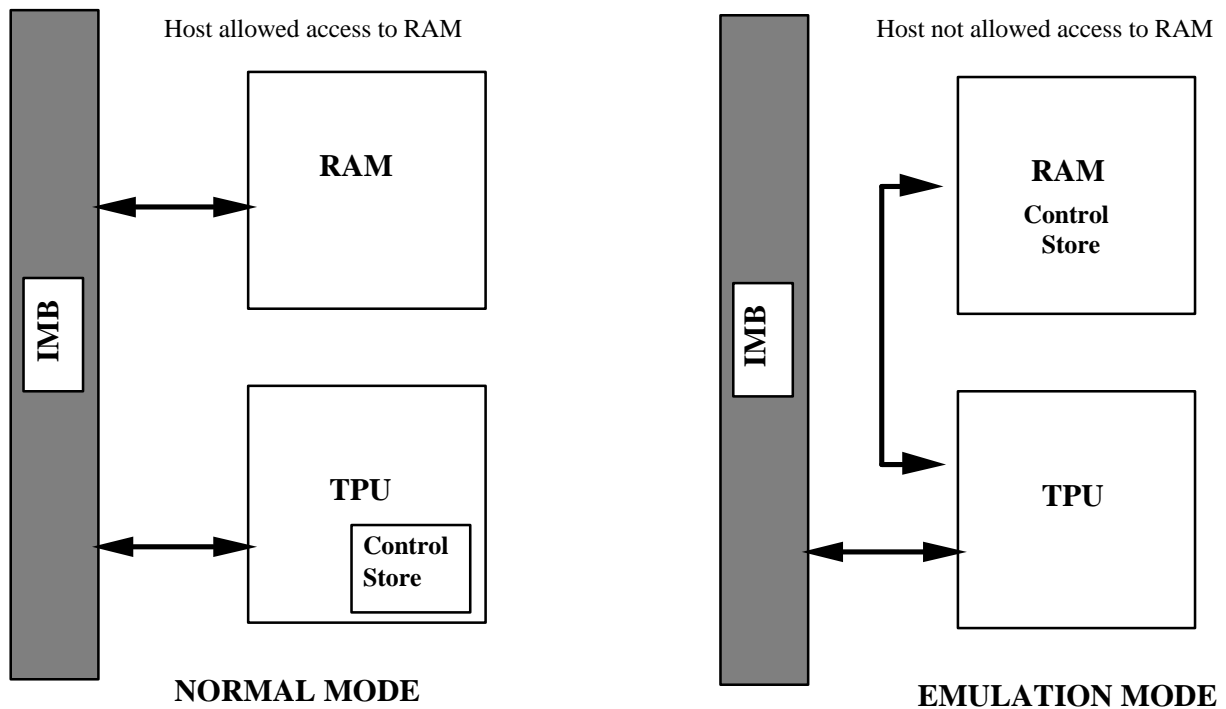
- Input Capture/Input Transition Counter
- Output Compare
- Pulse-Width Modulation
- Synchronized Pulse-Width Modulation
- Period Measurement with Additional Transition Detect
- Period Measurement with Missing Transition Detect
- Position-Synchronized Pulse Generator
- Stepper Motor
- Period/Pulse-Width Accumulator

New Mask:

- New TPU function set aimed at Motor and Motion Control.
- Supports multi axis control in some systems.
- Supports the following motor types:-
 - D.C. Brushed & brushless, A.C. Induction
 - Stepper, Switched Reluctance.
- Quadrature Decode
- Multi channel PWM
- Hall Effect Decode
- Multi phase motor commutation
- Input Capture/Transition counter
- Queued Output Match
- Pulse time accumulator (New PPWA)
- Stepper motor
- Asynchronous Receiver/transmitter
- Frequency Measurement

Emulation Capability

- Emulation capability is provided so the user can create new time functions.
- To use emulation mode, the user must first load microcode into RAM via the IMB, then assert the EMU bit in TPU module configuration register.



TPU Function Library

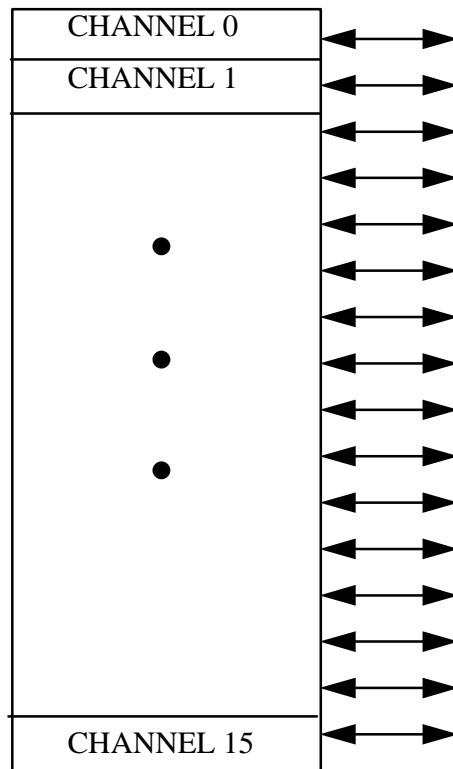
- A **collection of debugged and documented TPU functions**, including those in the standard ROM.

- library will reside on **MCU bulletin board** (512)891-FREE (3733).

- Any customer can **select a new function set** from those available and run the TPU from **emulation RAM**.
- A **custom ROM** will be available subject to Minimum Order Quantity and mask charge.
- **Master Library Guide** available. Covers compatibility issues that are important for anyone combining **custom functions** with **library** or **standard functions**.
- **To order** a Master Library Guide call the Motorola Literature Distribution Center at 602-994-6561 and ask for Application Note #TPUPN00-D.

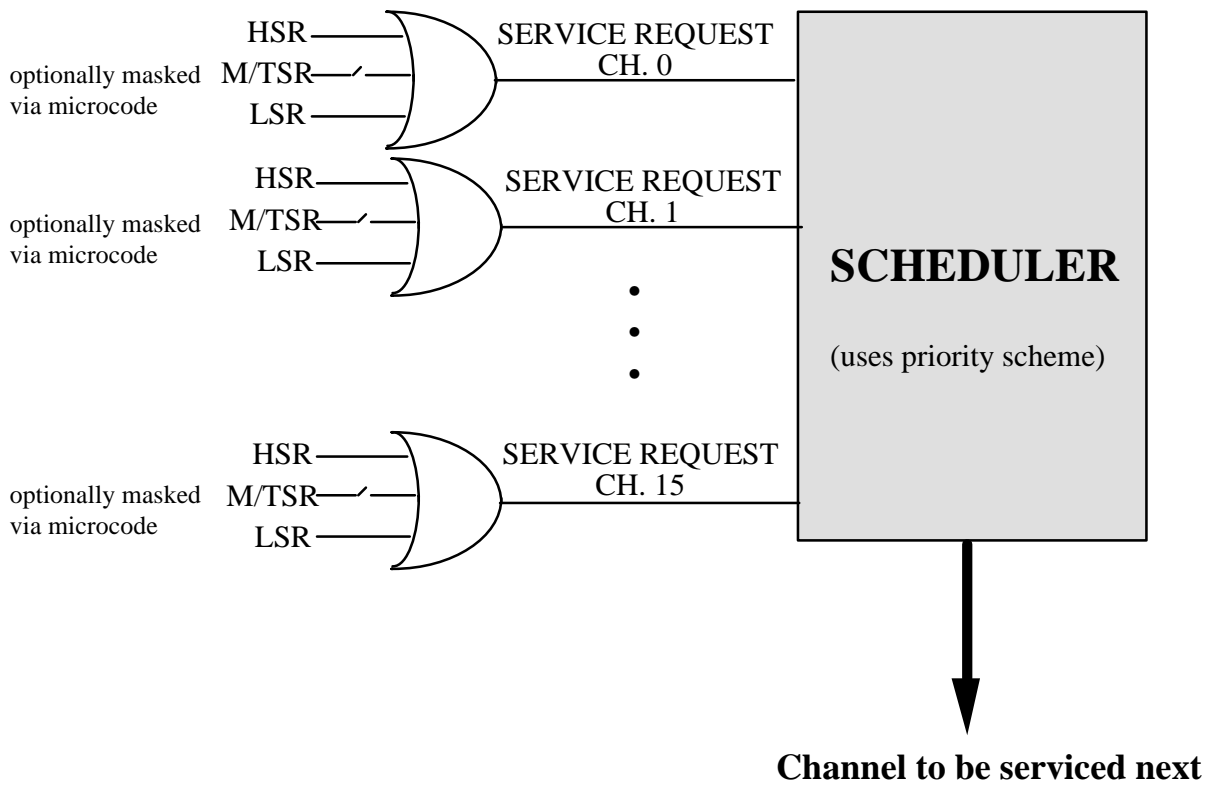
TPU Features

- **16 input/output pins -- each associated with a timer channel**



- Each channel can perform any time function.
- Each time function may be assigned to more than one channel at a given time.
- A priority scheme gives frequent service to high priority channels and minimum service to low priority channels.

Scheduler



HSR - Host Service Request (from CPU)
M/TSR - Match or Transition Service Request (from Channel hardware)
LSR - Link Service Request (from Execution Unit)

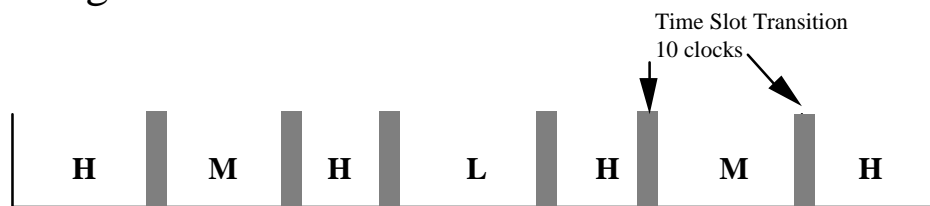
• **Within a channel, HSRs have priority over M/TSRs and LSRs.**

Priority Scheme

LEVEL 1 (PRIMARY)

Each channel assigned a priority level of high, middle, or low via Channel Priority Register.

Scheduler assigns TIME SLOTS as follows



TIME SLOTS vary in length depending on executing state

HIGH PRIORITY given 4 time slots

MIDDLE PRIORITY given 2 time slots

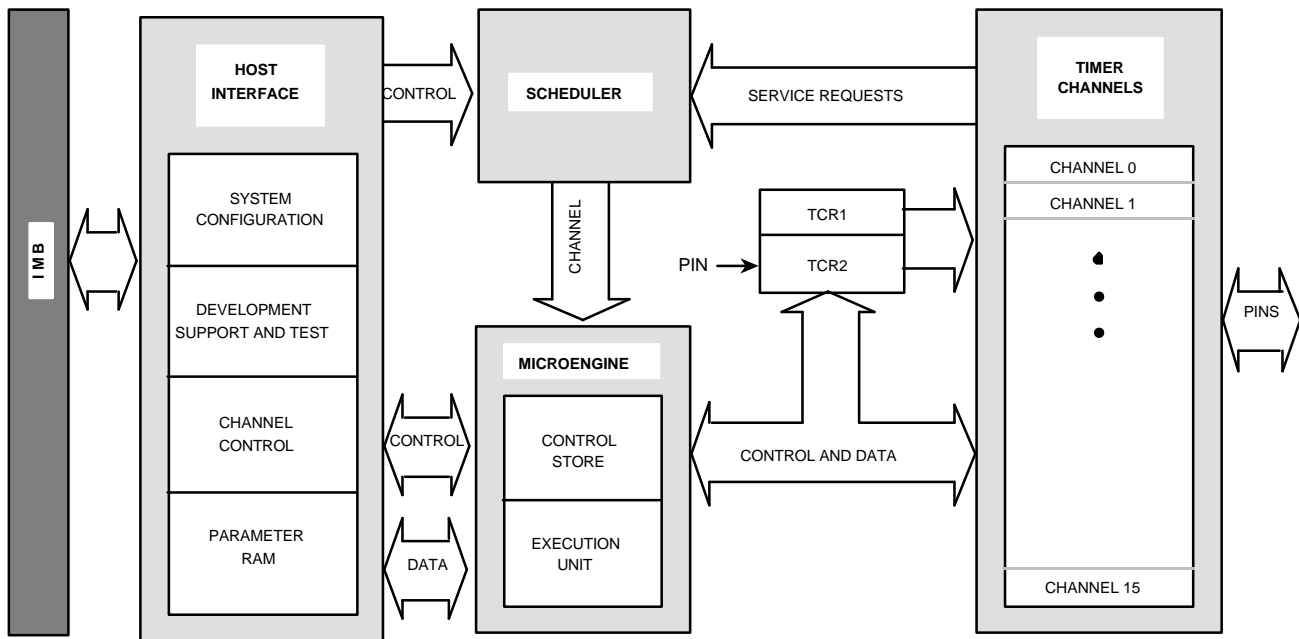
LOW PRIORITY given 1 time slot

LEVEL 2 (SECONDARY)

If 2 channels that have the same priority level request service at the same time, the lowest numbered channel has priority.

(Note: a Round Robin scheme ensures that every channel will be serviced.)

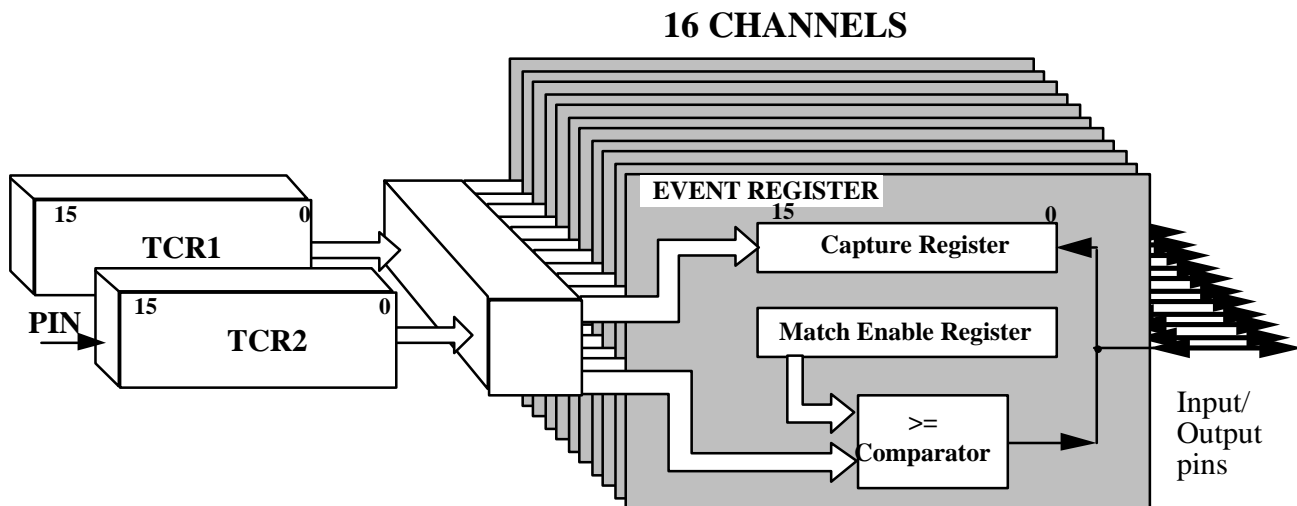
TPU Simplified Block Diagram



Channel Features

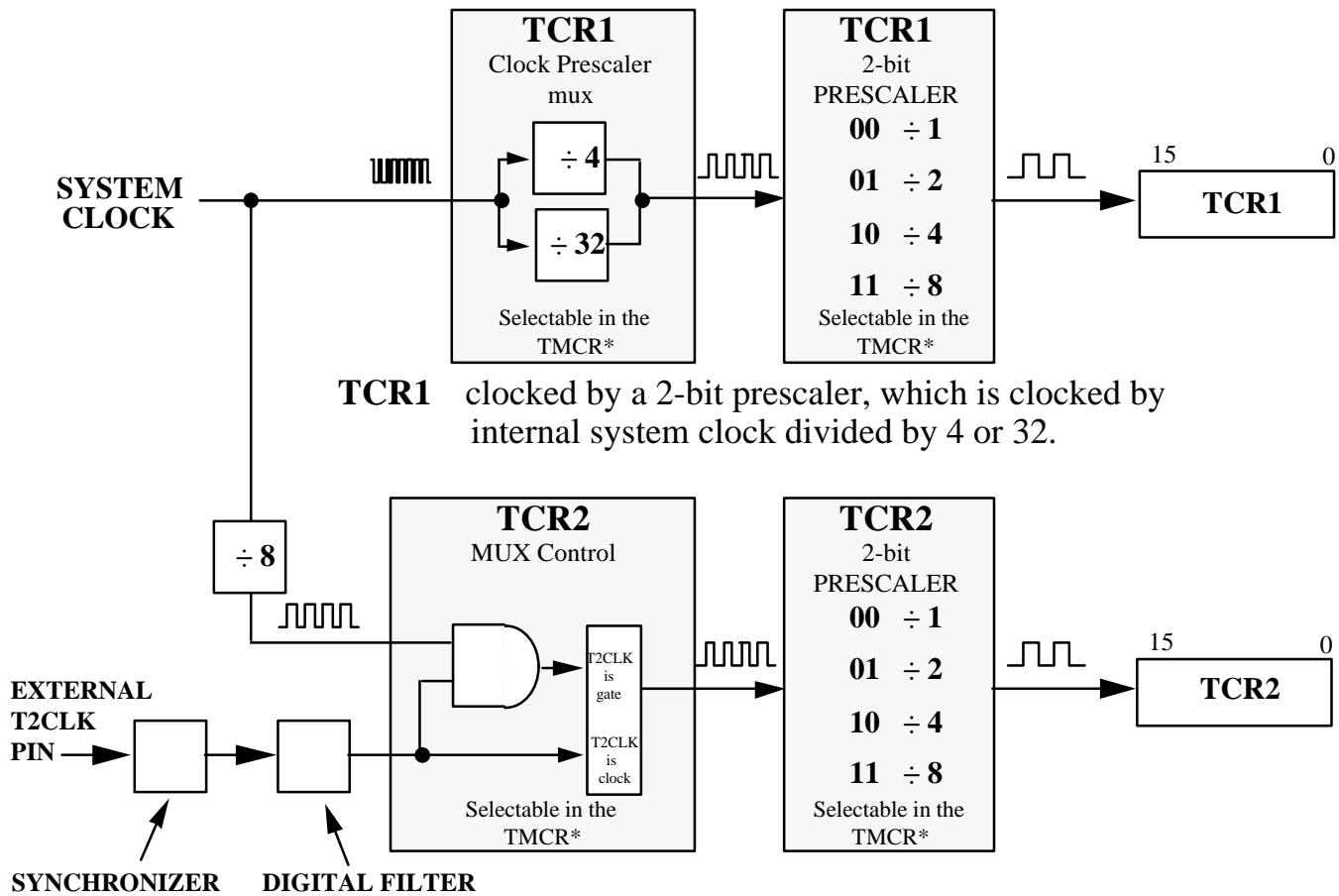
Each channel has its own pin and **Event Register**. The Event Register consists of:

- 16-bit Match Enable Register
- 16-bit Capture Register
- 16-bit Greater-than-or-equal-to Comparator



- **Each** channel can be synchronized to one or both of the two 16-bit free-running Timer Count Registers (**TCR1** and **TCR2**).
- This is done in **microcode**. (The standard functions allow the user to choose the time base via CHANNEL_CONTROL (Parameter 0) in Parameter RAM. The microcode reads this parameter and sets the time base accordingly .)
- Setting up for **Match** and **Transition events** is also done in **microcode**.

TCR1 and TCR2 CLOCKING



TCR1 clocked by a 2-bit prescaler, which is clocked by internal system clock divided by 4 or 32.

TCR2 clocked by 2-bit prescaler which is clocked in one of two ways:

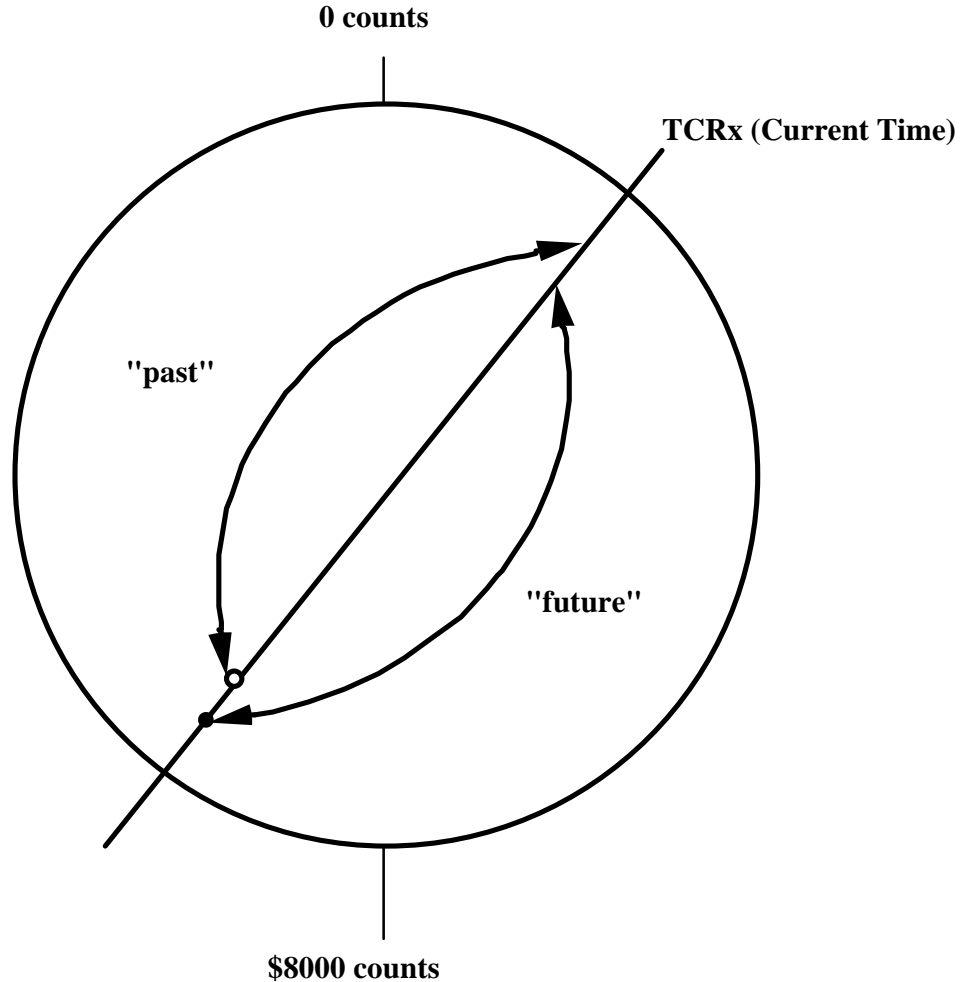
- 1) by the system clock ÷ 8 (TCR2 pin used as gate)
- 2) by the external TCR2 pin (TCR2 pin used as clock)

NOTE: Minimum resolution @ 16MHz: TCR1 - 240ns/TIC
TCR2 - 500ns/TIC

* **TMCR** - TPU Module Configuration Register

Past and Future

- **Current TCR + \$8000 is the MAXIMUM future value**



COMPARATOR LOGIC:

If the sum of the match register and the one's complement of the match TCR time results in bit 15 equaling one, then the current match TCR time is greater than or equal to the match register.

eg. Match Register = \$012C

If TCR=\$012B (1's complement =\$FED4)

If TCR=\$012C (1's complement =\$FED3)

If TCR=\$012D (1's complement =\$FED2)

012C	012C	012C
<u>+FED4</u>	<u>+FED3</u>	<u>+FED2</u>
0000	FFFF	FFFE
msb = 0	msb = 1	msb = 1

TEMPUS FUGIT

TPU Register Map

CONTROL REGISTERS

PARAMETER RAM

	15			0
S	\$YFFE00	MODULE CONFIGURATION REGISTER		
S	\$YFFE02	MODULE TEST REGISTER		
S	\$YFFE04	DEVELOPMENT SUPPORT		
		CONTROL & STATUS REGISTERS		
S	\$YFFE08	INTERRUPT CONFIGURATION REGISTER		
S	\$YFFE0A	CH 15	CH 14	CH 13
S	\$YFFE0C	CH 11	CHANNEL FUNCTION SELECT REGISTERS (4 bits per channel)	
		CH 7	CH 2	CH 1
		CH 3	CH 12	CH 8
		CH 15	HOST SEQUENCE REGISTERS (2 bits per channel)	
X	\$YFFE14	CH 7	CH 8	CH 0
X	\$YFFE18	CH 15	HOST SERVICE REQUEST REGISTERS (2 bits per channel)	
		CH 7	CH 8	CH 0
S	\$YFFE1C	CH 15	CHANNEL PRIORITY REGISTERS (2 bits per channel)	
		CH 7	CH 8	CH 0
S	\$YFFE20	CH 15	CH 14	CH 13
S	\$YFFE22	LINK REGISTER		
S	\$YFFE24	SERVICE GRANT LATCH REGISTER		
S	\$YFFE26	DECODED CHANNEL NUMBER REGISTER		
S	\$YFFE28	RESERVED		
S	\$YFFEF0	RESERVED		
X	\$YFFF00	CH0 PARAM 0		
X	\$YFFF02	CH0 PARAM 1		
X	\$YFFF04	CH0 PARAM 2		
X	\$YFFF06	CH0 PARAM 3		
X	\$YFFF08	CH0 PARAM 4		
X	\$YFFF0A	CH0 PARAM 5		
		NOT IMPLEMENTED		
		...		
X	\$YFFFE0	CH14 PARAM 0		
X	\$YFFFE2	CH14 PARAM 1		
X	\$YFFFE4	CH14 PARAM 2		
X	\$YFFFE6	CH14 PARAM 3		
X	\$YFFFE8	CH14 PARAM 4		
X	\$YFFFEA	CH14 PARAM 5		
X	\$YFFFE C	CH14 PARAM 6		
X	\$YFFFE E	CH14 PARAM 7		
X	\$YFFF00	CH15 PARAM 0		
X	\$YFFF02	CH15 PARAM 1		
X	\$YFFF04	CH15 PARAM 2		
X	\$YFFF06	CH15 PARAM 3		
X	\$YFFF08	CH15 PARAM 4		
X	\$YFFF0A	CH15 PARAM 5		
X	\$YFFF0C	CH15 PARAM 6		
X	\$YFFF0E	CH15 PARAM 7		

NOTE: Except for CISR, TPU Registers & Parameter RAM are not byte-accessible.

Y = Programmable as \$7 or \$F
 S = Accessible only in Supervisor mode.
 X = Programmable with SUPV bit as Unrestricted or Supervisor-only accessible.

- Development support or test registers

ALL PARAMETER LOCATIONS are programmable as either Supervisor-only accessible or Unrestricted by the SUPV bit in the TMCr

CHANNEL 0

CHANNELS 0-13 have 6 parameters

CHANNEL 14

CHANNELS 14-15 have 8 parameters.

CHANNEL 15

Host Service & Host Sequence Registers

HOST SERVICE REGISTER (HSR) CPU request service of function. Usually for initialization; also to request or give update.

2 bit field per channel:



HOST SEQUENCE REGISTER (HSQ) Further qualifies Host Service Register.

2 bit field per channel:



- To issue a Host Service Request, **CPU writes** to the Host Service Register (HSR). The **TPU hardware clears** the Host Service Request field after service of the state.

Eg. **Output Compare (OC):**

Host Service Request Code: Host Sequence Code:

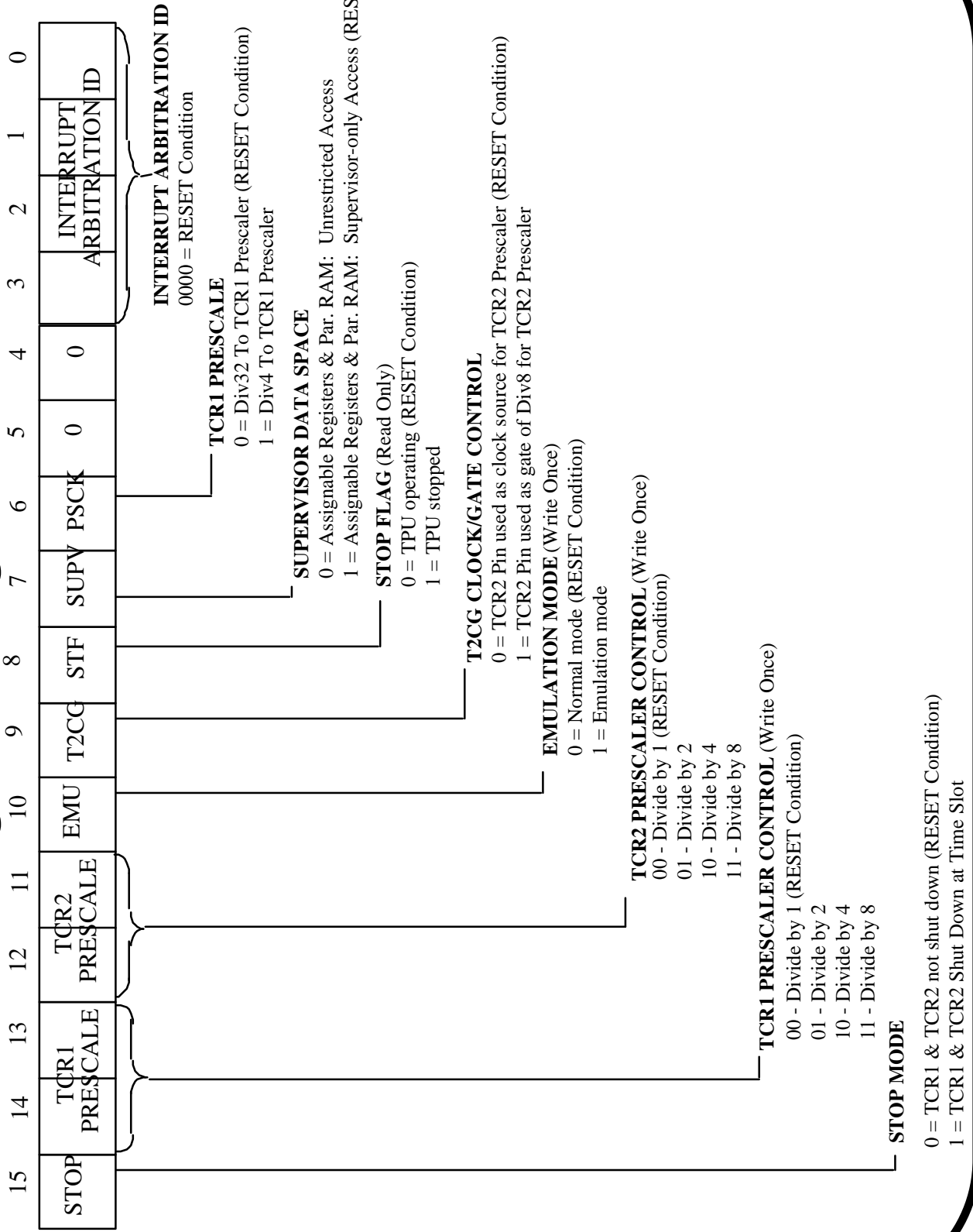
00 = None	0X = If Host-Init mode, execute all code in	01
= Host-Initiated Pulse Mode function		
= If Host-Init mode, execute only code that updates TCRn Params	10 = (Not Implemented)	1X
	11 = Initialization, Continuous Pulse Mode	

Procedure: When developing a function, the microcode programmer specifies the meaning of HSRs and HSQs and documents this for the CPU programmer.

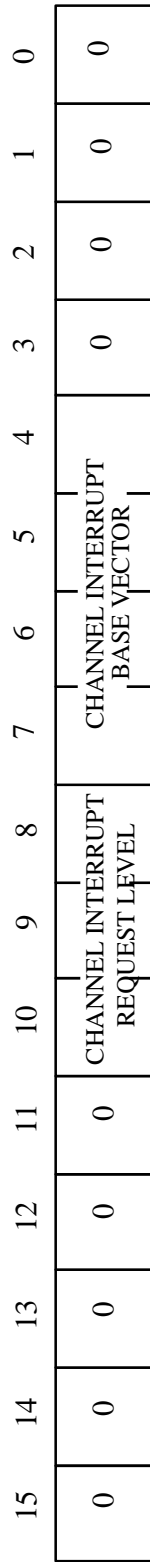
TPU Configuration

- 1) Determine the overall operating characteristics of the TPU by writing to the **Module Configuration Register**
- 2) Determine the TPU Interrupt structure by writing to the **Interrupt Configuration Register**

TPU Module Configuration Register (TMCR)



TPU Interrupt Configuration Register (TICR)



CHANNEL INTERRUPT BASE VECTOR

Forms upper 1/2 of Vector # for Channel Interrupt; Channel # forms lower 1/2
0000 = RESET Condition

INTERRUPT REQUEST LEVEL (for all channels)

0 = Interrupts Disabled (RESET Condition)
1-7 = IRQ1-7 Respectively

EX. A \$4 IN CHANNEL INTERRUPT BASE VECTOR FIELD MEANS:

Interrupt on Channel 0 goes to Vector \$40 on Exception Vector Table
 " Channel 1 " " Vector \$41

.

" Channel 15 " Vector \$4F "

Channel Interrupt Enable Register (CIER)

\$YFFE0A

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

CHAN 15	CHAN 14	CHAN 13	CHAN 12	CHAN 11	CHAN 10	CHAN 9	CHAN 8	CHAN 7	CHAN 6	CHAN 5	CHAN 4	CHAN 3	CHAN 2	CHAN 1	CHAN 0
------------	------------	------------	------------	------------	------------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------

RESET: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 = Channel Interrupts disabled

1 = Channel Interrupts enabled

Channel Interrupt Status Register(CISR)

\$YFFE20

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

CHAN 15	CHAN 14	CHAN 13	CHAN 12	CHAN 11	CHAN 10	CHAN 9	CHAN 8	CHAN 7	CHAN 6	CHAN 5	CHAN 4	CHAN 3	CHAN 2	CHAN 1	CHAN 0
------------	------------	------------	------------	------------	------------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------

RESET: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

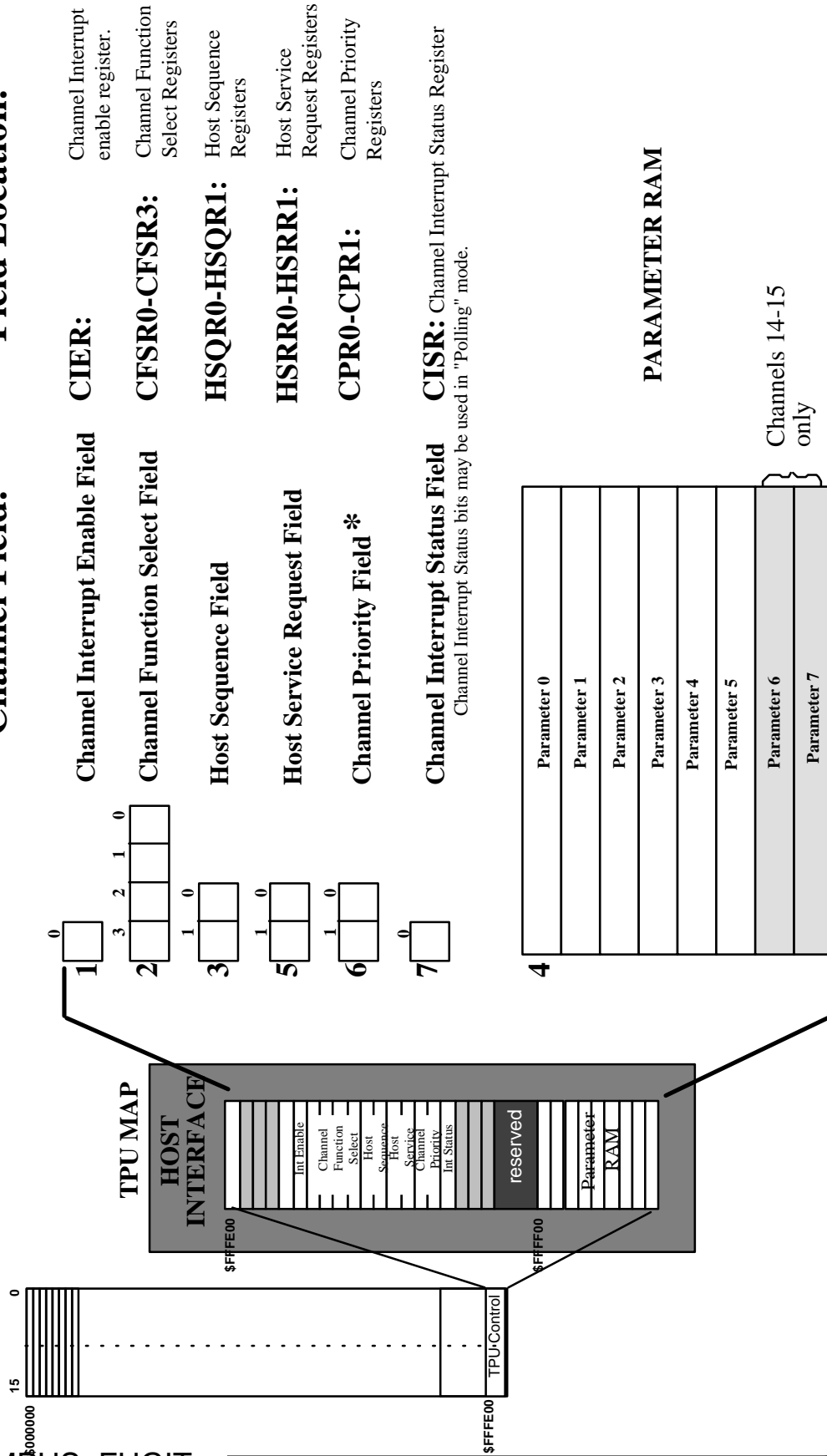
To negate a channel interrupt flag: read the flag and write back a zero.
If a new interrupt occurs between the read and write, the flag remains asserted.

Channel Configuration

- 1) Enable interrupts for appropriate channels (CHANNEL INTERRUPT ENABLE REG)
- 2) Select time functions for channels (CHANNEL FUNCTION SELECT REG)
- 3) Choose operating options for time functions (HOST SEQUENCE REG)
- 4) Write pertinent parameters for configured channels (PARAMETER RAM)
- 5) Issue Host Service Requests to initialize the active channels (HOST SERVICE REQUEST REG)
- 6) Set channel priorities to Low, Middle, or High (CHANNEL PRIORITY REG)
- 7) Monitor Host Service Request Registers (or a channel interrupt) for completion of initialization (HOST SERVICE REQUEST REG)

MAP for EACH TPU CHANNEL

TEMPUS FUGIT



Channel Field: Field Location:

Channel Interrupt Enable Field	CIER:	Channel Interrupt enable register.
Channel Function Select Field	CFSR0-CFSR3:	Channel Function Select Registers
Host Sequence Field	HSQR0-HSQR1:	Host Sequence Registers
Host Service Request Field	HSRR0-HSRR1:	Host Service Request Registers
Channel Priority Field *	CPR0-CPR1:	Channel Priority Registers

Channel Interrupt Status Field **CISR:** Channel Interrupt Status Register
 Channel Interrupt Status bits may be used in "Polling" mode.

PARAMETER RAM

Channels 14-15 only

* **Last in Configuration Sequence** — Enables Channel for Scheduling
 (Nothing occurs until channel is serviced by TPU μ -engine.)

FUNCT. TABLE

TEMPUS FUGIT

FUNCTION NAME	FUNCTION CODE	HOST SERVICE REQUEST CODE	HOST SEQUENCE CODE
DIO Discrete Input/Output	\$8	0 = None 1 = Force Output High 2 = Force Output Low 3 = Initialization, Input Specified 3 = Initialization, Periodic Input 3 = Update Pin Status Parameter	0 0 = Trans Mode - Record Pin on Transition 0 = Trans Mode - Record Pin on Transition 0 = Trans Mode - Record Pin on Transition 1 = Match Mode - Record Pin at MATCH_RATE 2 = Record Pin State on HSR 11
ITC Input Capture/ Input Transition Counter	\$A	0 = None 1 = Initialization 2 = (Not Implemented) 3 = (Not Implemented)	0 = No Link, Single Mode 1 = No Link, Continuous Mode 2 = Link, Single Mode 3 = Link, Continuous Mode
OC Output Compare	\$E	0 = None 1 = Host-Initiated Pulse Mode 2 = (Not Implemented) 3 = Continuous Pulse Mode	0 = If HSR=01, Execute All Code in Function 1 = If HSR=01, Execute All Code in Function 2 = If HSR=01, Only Update TCRn Parameters 3 = If HSR=01, Only Update TCRn Parameters
PWM Pulse Width Modulation	\$9	0 = None 1 = Immediate Update Request 2 = Initialization 3 = (Not Implemented)	(None Implemented)
SPWM Synchronized Pulse Width Modulation	\$7	0 = None 1 = (Not Implemented) 2 = Initialization 3 = Immediate Update Request	0 = Mode 0 1 = Mode 1 2 = Mode 2 3 = (Not Implemented)
PMA/PMU Period Measurement with Additional/ Missing Transition Detect	\$B	0 = None 1 = Initialization 2 = (Not Implemented) 3 = (Not Implemented)	0 = PMA Bank Mode 1 = PMA Count Mode 2 = PMM Bank Mode 3 = PMM Count Mode
PSP Position-Synchronized Pulse Generator	\$C	0 = None 1 = Immediate Update Request 2 = Initialization 3 = Force Change	0 = Pulse Width Set by Angle 1 = Pulse Width Set by Time 2 = Pulse Width Set by Angle 3 = Pulse Width Set by Time
SM Stepper Motor	\$D	0 = None 1 = None 2 = Initialization 3 = Step Request	(None Implemented)
PPWA Period/Pulse-Width Accumulator	\$F	0 = None 1 = (Not Implemented) 2 = Initialization 3 = (Not Implemented)	0 = 24-Bit Period 1 = 16-Bit Period + Link 2 = 24-Bit Period Width 3 = 16-Bit Pulse Width + Link

Example CPU32 Init Code from Reset

EXAMPLE -- CODE TO INITIALIZE THE TPU TO RUN INPUT TRANSITION COUNT (ITC) ON CHANNEL 1 AND PULSE WIDTH MODULATION (PWM) ON CHANNEL 0:

****Initialize TPU Module Configuration Register and TPU Interrupt Control Register to set up for interrupts from TPU and to set up for a fast clock.****

```
MOVE.W  #$004E, TMCR      ;prescale TCR1 by 4, set IARB to $E
MOVE.W  #$0640, TICR      ;TPU interrupt level = 6, vectors $4X
```

****Enable interrupts on Channels 0 & 1 only by setting corresponding bits in Channel Interrupt Enable Register****

```
MOVE.W  #$0003, CIER      ;enable interrupts for Ch. 0 & 1
```

****Choose ITC for Channel 1 and PWM for Channel 0 by writing to the Channel Function Select Register****

```
MOVE.W  #$00A9, CFSR3     ;ITC ($A) to Ch. 1, PWM (9) to Ch. 0
```

****Choose options for Channel 1 by writing to the Host Sequence Register. Set up parameters for Channel 1 by writing to Channel 1's parameters.****

```
MOVE.W  #$0004, HSQR1     ;No Link, Cont. mode (%01) to Ch. 1
MOVE.W  #$0007, CH1PAR0   ;Capture TCR1 on rising edges
MOVE.W  #$000E, CH1PAR1   ;Bank address pointed to nonexistent addr.
MOVE.W  #$000A, CH1PAR2   ;Max count = $A for Ch. 1
```

****Set up parameters for Channel 0 by writing to Channel 0's parameters.****

```
MOVE.W  #$0091, CH0PAR0   ;Match TCR1, initialize pin high
MOVE.W  #$1000, CH0PAR2   ;High time = $1000 TCR1 tics
MOVE.W  #$2000, CH0PAR3   ;Period = $2000 TCR1 tics
```

****Request service to initialize Channels 1 and 0 by writing to the Host Service Register****

```
MOVE.W  #$0006, HSRR1     ;Host Service Requests to init. Ch. 0 & 1
                          ;(%01 for Ch. 1 and %10 for Ch. 0)
```

****Assign priorities for Channels 1 and 0 by writing to the Channel Priority Register.****

```
MOVE.W  #$000B, CPR1      ;Ch. 1 = middle priority(%10)
                          ;Ch. 0 = high priority(%11)
```

****Make sure Channels 1 and 0 have been initialized before continuing. (This may not be necessary.)****

```
WAIT:
MOVE.W  HSRR1, D0          ;Check Host Service bits for Ch. 1 & 0
ORI.B   #$0F, D0          ;if Host Service Request bits =00 then
                          ;channel has been serviced.
BNE     WAIT
```

CPU16 INIT CODE from RESET

EXAMPLE -- CODE TO INITIALIZE THE TPU TO RUN INPUT TRANSITION COUNT (ITC) ON CHANNEL 1 AND PULSE WIDTH MODULATION (PWM) ON CHANNEL 0:

****Initialize TPU Module Configuration Register and TPU Interrupt Control Register to set up for interrupts from TPU and to set up for a fast clock.****

```
LDD  #$004E
STD  TMCR      ;prescale TCR1 by 4, set IARB to $E
LDD  #$0640
STD  TICR      ;TPU interrupt level = 6, vectors $4X
```

****Enable interrupts on Channels 0 & 1 only by setting corresponding bits in Channel Interrupt Enable Register****

```
LDD  #$0003
STD  CIER      ;enable interrupts for Ch. 0 & 1
```

****Choose ITC for Channel 1 and PWM for Channel 0 by writing to the Channel Function Select Register****

```
LDD  #$00A9
STD  CFSR3     ;ITC ($A) to Ch. 1, PWM (9) to Ch. 0
```

****Choose options for Channel 1 by writing to the Host Sequence Register. Set up parameters for Channel 1 by writing to Channel 1's parameters.****

```
LDD  #$0004
STD  HSQR1     ;No Link, Cont. mode (%01) to Ch. 1
LDD  #$0007
STD  CH1PAR0   ;Capture TCR1 on rising edges
LDD  #$000E
STD  CH1PAR1   ;Bank address pointed to nonexistent addr.
LDD  #$000A
STD  CH1PAR2   ;Max count = $A for Ch. 1
```

****Set up parameters for Channel 0 by writing to Channel 0's parameters.****

```
LDD  #$0091
STD  CH0PAR0   ;Match TCR1, initialize pin high
LDD  #$1000
STD  CH0PAR2   ;High time = $1000 TCR1 tics
LDD  #$2000
STD  CH0PAR3   ;Period = $2000 TCR1 tics
```

****Request service to initialize Channels 1 and 0 by writing to the Host Service Register****

```
LDD  #$0006
STD  HSRR1     ;Host Service Requests to init. Ch. 0 & 1
STD  HSRR1     ;(%01 for Ch. 1 and %10 for Ch. 0)
```

CPU16 INIT CODE continued

****Assign priorities for Channels 1 and 0 by writing to the Channel Priority Register.****

```
LDD  #$000B    ;Ch. 1 = middle priority(%10)
STD  CPR1      ;Ch. 0 = high priority(%11)
```

****Make sure Channels 1 and 0 have been initialized before continuing. (This may not be necessary.)****

```
WAIT:
LDD  HSRR1     ;Check Host Service bits for Ch. 1 & 0
ANDD #$000F    ;if Host Service Request bits = 00 then
                ;channel has been serviced
BNE  WAIT
```

TPU MICROCODE OVERVIEW

Instruction Format

- 1 MICROINSTRUCTION = 32 bits
- Executes in 2 CPU clocks (regardless of TCR1 & TCR2 speeds)



MICROINSTRUCTION made of SUBCOMMANDS:

Subcommand	Subcommand Designator
Channel Control	chan
Execution Unit	au
RAM	ram
Microengine/Sequencing	—

SUBCOMMANDS made of FIELDS.

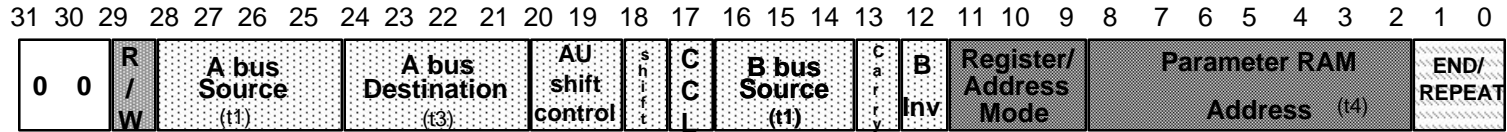
- MICROINSTRUCTIONS separated by "."
- SUBCOMMANDS separated by ";"
- FIELDS separated by ","

EX. chan field,
 field;
 au field, field;

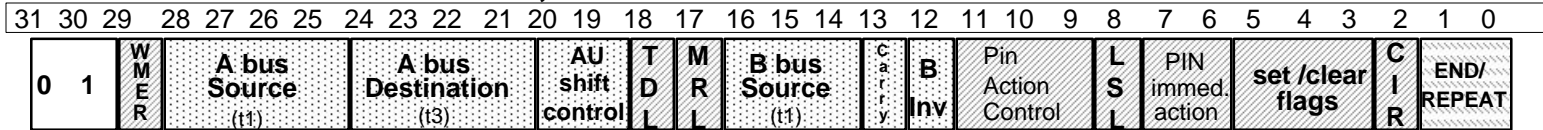
 ram field.
 chan field, field;
 ram field;
 goto label.

INSTRUCTION FORMATS

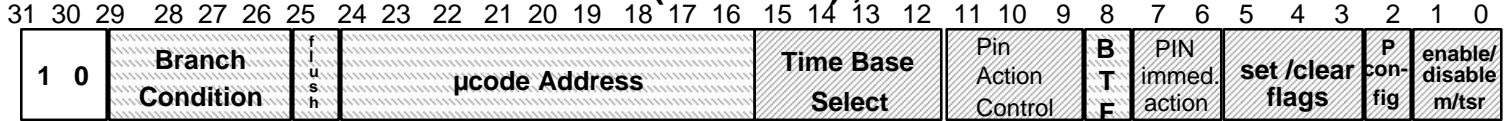
FORMAT 1: EXECUTION UNIT AND RAM



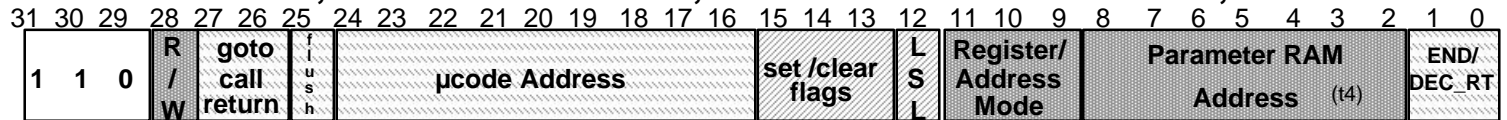
FORMAT 2: EXECUTION UNIT, FLAG AND CHANNEL CONTROL



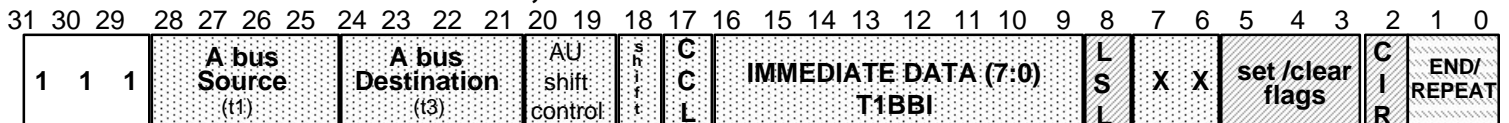
FORMAT 3: CONDITIONAL BRANCH (IF...THEN) , FLAG AND CHANNEL CONTROL



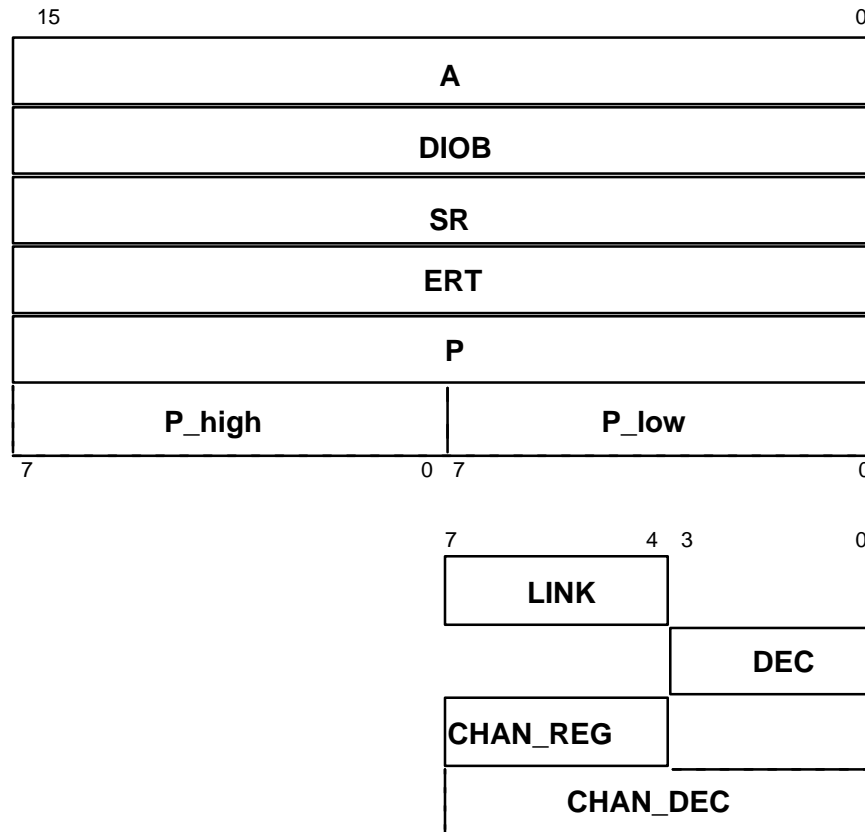
FORMAT 4: GOTO, CALL SUBROUTINE, RETURN from SUBROUTINE, FLAG AND RAM



FORMAT 5: EXECUTION UNIT, IMMEDIATE DATA AND FLAG



TPU Registers



NOTE: There is only ONE of each register pictured. All channels share these registers.

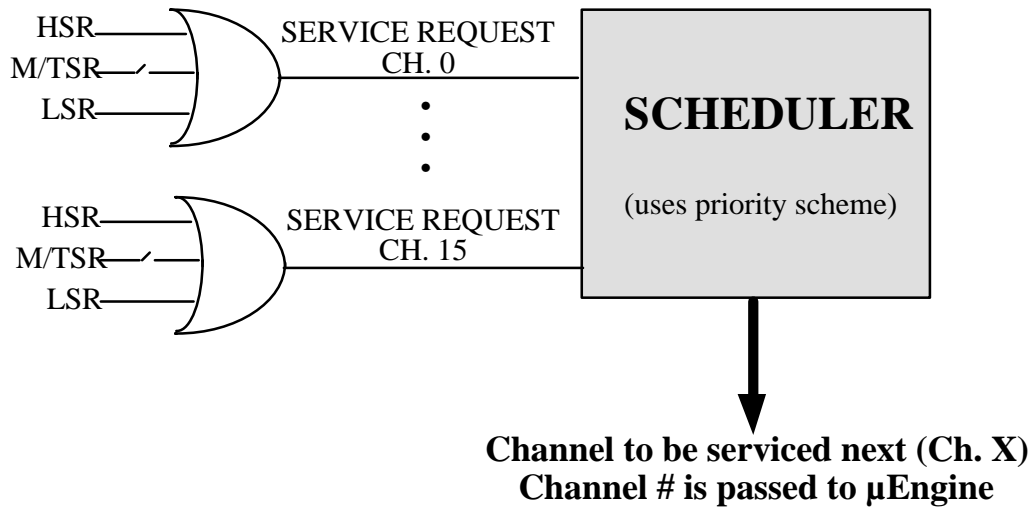
Name summary: A=Accumulator, DIOB= Data I/O Buffer, SR = Shift Register, ERT = Event Register Temporary, P = Preload Register, LINK= Link register, DEC=Decrementor Register, CHAN_REG=Channel Register

Note: All channels share these registers. Each channel has its own MER (Match Event Register) and Capture Register.

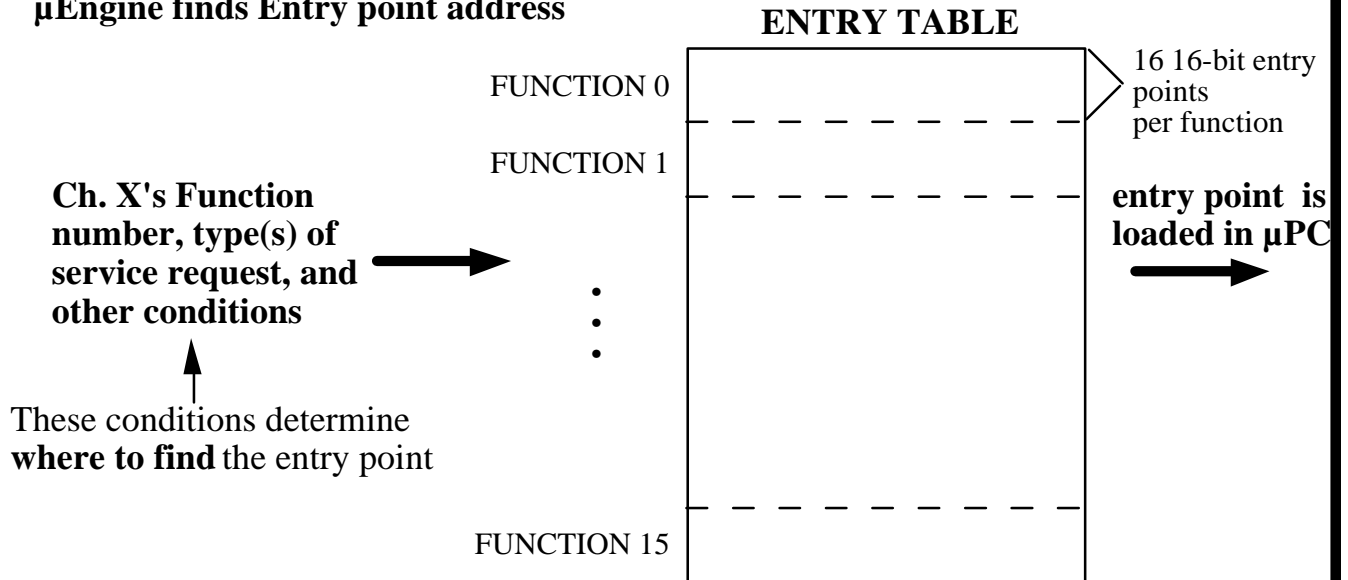
ENTRY POINTS

Entry Table Overview

DURING TIME SLOT TRANSITION:



μEngine finds Entry point address



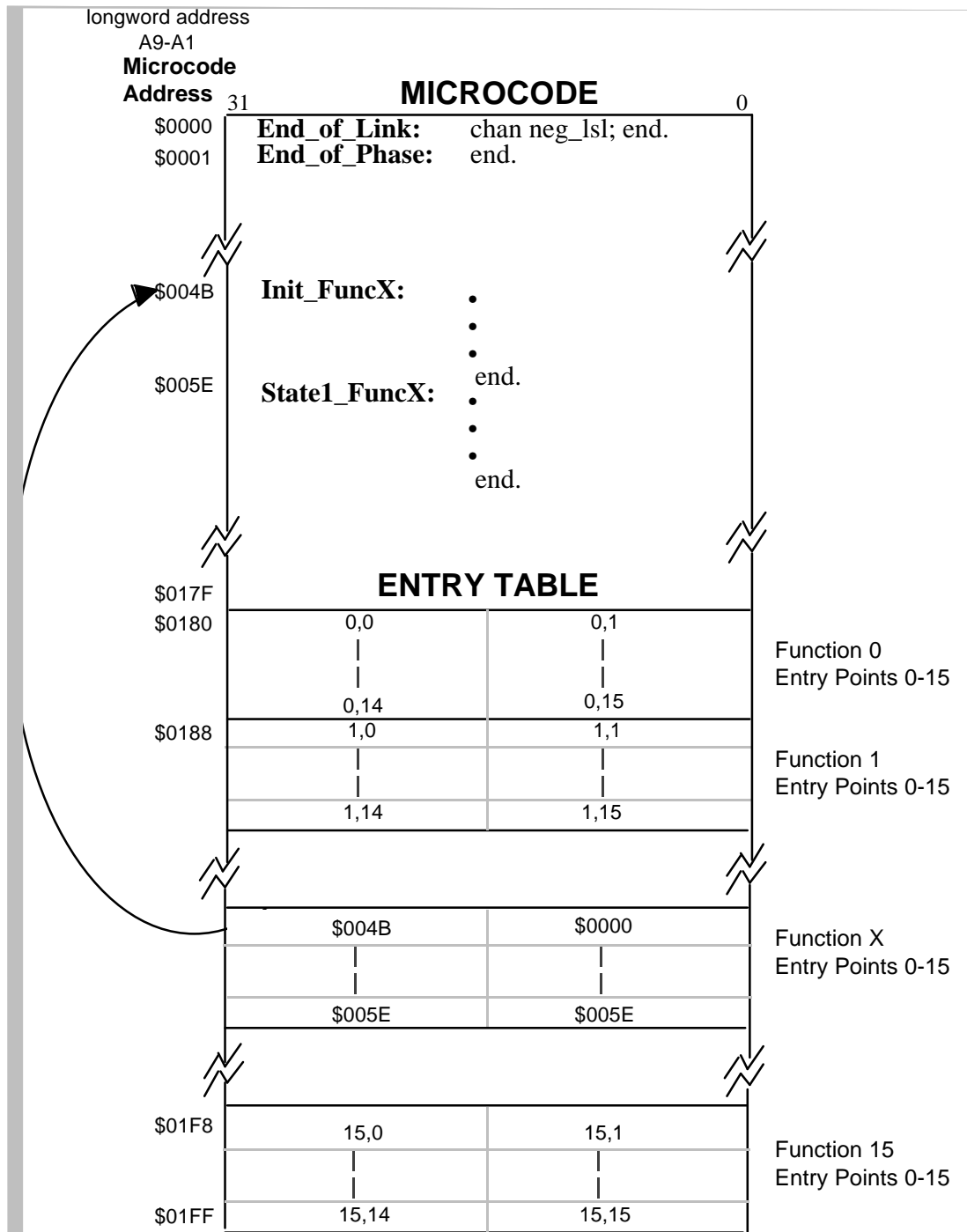
Entry Table Lookup

Entry Points	Service Request Sources			Channel Conditions	
	Host Request (HSR)	Link Request (LSR)	Match/Transition Service Request (M/TSR)	Pin State	Channel Flag 0
0	01	x	x	0	x
1	01	x	x	1	x
2	10	x	x	x	x
3	11	x	x	x	x
4	00	0	1	0	0
5	00	0	1	0	1
6	00	0	1	1	0
7	00	0	1	1	1
8	00	1	0	0	0
9	00	1	0	0	1
10	00	1	0	1	0
11	00	1	0	1	1
12	00	1	1	0	0
13	00	1	1	0	1
14	00	1	1	1	0
15	00	1	1	1	1

- 4 HSR entry points
- 4 LSR entry points
- 4 M/TSR entry points
- 4 LSR and M/TSR entry points

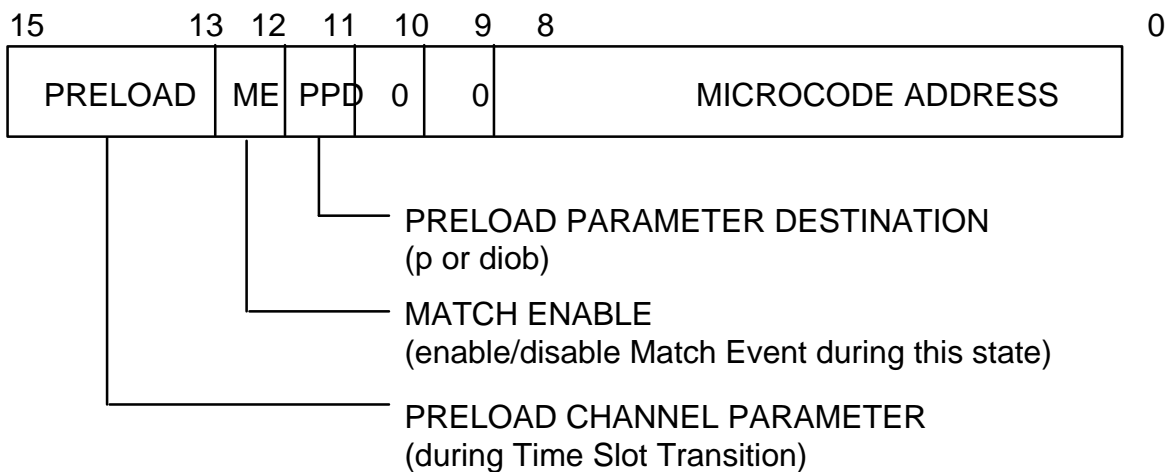
- Channel Flag 0 is controlled through microcode

Entry Table and Control Store



Note: Unused entry points should point to **End_of_Link** unless links are received in the function, in which case **End_of_Phase** should be used. This ensures that unwanted service requests are correctly terminated.

Entry Points



EG.

```
%entry name = INIT_PWM; start_addr INIT_PWM; disable_match;
cond hsr1=1, hsr0=1, lsr=x, m/tsr=x, flag0=x , pin=x;
ram p<-@PAR_HIGHTIME.
```

INIT_PWM:

```
chan tbs:=out_m1_c1,
      pin:=high,
      pac:=low.
      .
      .
      .
end.
```

EG.

```
%entry name = MATCHL_PWM; start_addr *; disable_match;
cond hsr1=0, hsr0=0, lsr=x, m/tsr=1, flag0=x, pin=LOW;
ram diob<-prm0.
```

```
au ert := ert +diob.
    .
    .
end.
```

Entry Point Exercises (1/2)

1. A Link requests service on a channel and before the Scheduler can respond to the request the Host issues an HSR1=0, HSR0=1. Which entry point will be selected, the Host entry point or the Link entry point?

2. How many entry points will be generated by the following directive?

```
%entry name = Exercise_5; start_addr *; disable_match;  
cond hsr1 = 0, hsr0 = 0, lsr = x, m/tsr = 1, pin = x, flag0 = x;  
ram p <- prm0.
```

3. Give the entry directive to load diob with parameter 1, disable match recognition, and start execution at the label "Detroit" when a match occurs with the pin low and flag0 set.

```
%entry name= _____; start_address _____; disable_match;  
cond hsr1 = ___, hsr0 = ___, lsr = ___, m/tsr = ___, pin = ___, flag0 = ___;
```

Entry Point Exercises (2/2)

4. A design calls for a state labeled "Match_Box" to be executed whenever there is a match and no link, and another state "Link_Box" to be executed whenever there is a link or a link and a match. Write two entry directives that cover all cases.

```
%entry name= _____; start_address _____; disable_match;  
cond hsr1 = ___, hsr0 = ___, lsr = ___, m/tsr = ___, pin = ___, flag0 = ___;
```

```
%entry name= _____; start_address _____; disable_match;  
cond hsr1 = ___, hsr0 = ___, lsr = ___, m/tsr = ___, pin = ___, flag0 = ___;
```

```
%entry name= _____; start_address _____; disable_match;  
cond hsr1 = ___, hsr0 = ___, lsr = ___, m/tsr = ___, pin = ___, flag0 = ___;
```

5. A function uses the entry points: HSR1=1, HSR0=1; HSR1=1, HSR0=0; and HSR1=0, HSR0=0, M/TSR=1, LSR=X, pin =1, flag0=X. The function does not receive links. Write the entry directives to cover all unused entry points.

```
%entry name= _____; start_address _____; disable_match;  
cond hsr1 = ___, hsr0 = ___, lsr = ___, m/tsr = ___, pin = ___, flag0 = ___;
```

```
%entry name= _____; start_address _____; disable_match;  
cond hsr1 = ___, hsr0 = ___, lsr = ___, m/tsr = ___, pin = ___, flag0 = ___;
```

```
%entry name= _____; start_address _____; disable_match;  
cond hsr1 = ___, hsr0 = ___, lsr = ___, m/tsr = ___, pin = ___, flag0 = ___;
```

%entry name= _____; start_address _____; disable_match;
cond hsr1 = ____, hsr0 = ____, lsr = ____, m/tsr = ____, pin = ____, flag0 = ____;

%entry name= _____; start_address _____; disable_match;
cond hsr1 = ____, hsr0 = ____, lsr = ____, m/tsr = ____, pin = ____, flag0 = ____;

%entry name= _____; start_address _____; disable_match;
cond hsr1 = ____, hsr0 = ____, lsr = ____, m/tsr = ____, pin = ____, flag0 = ____;

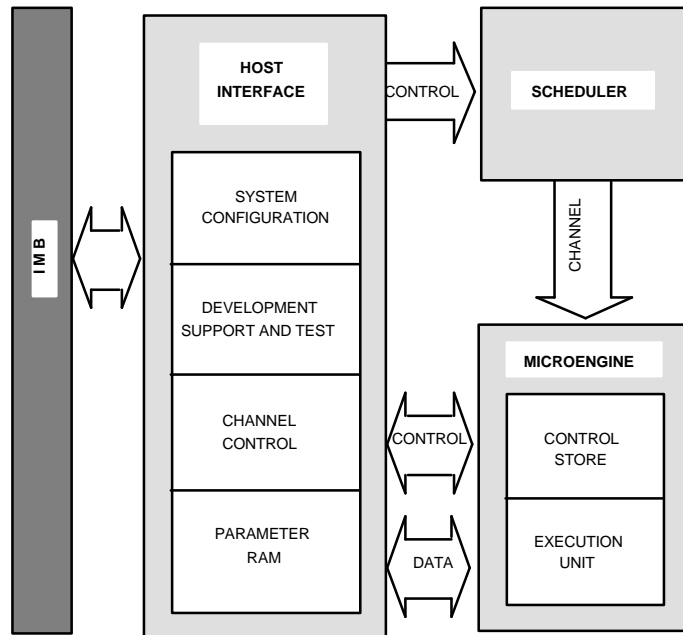
%entry name= _____; start_address _____; disable_match;
cond hsr1 = ____, hsr0 = ____, lsr = ____, m/tsr = ____, pin = ____, flag0 = ____;

%entry name= _____; start_address _____; disable_match;
cond hsr1 = ____, hsr0 = ____, lsr = ____, m/tsr = ____, pin = ____, flag0 = ____;

%entry name= _____; start_address _____; disable_match;
cond hsr1 = ____, hsr0 = ____, lsr = ____, m/tsr = ____, pin = ____, flag0 = ____;

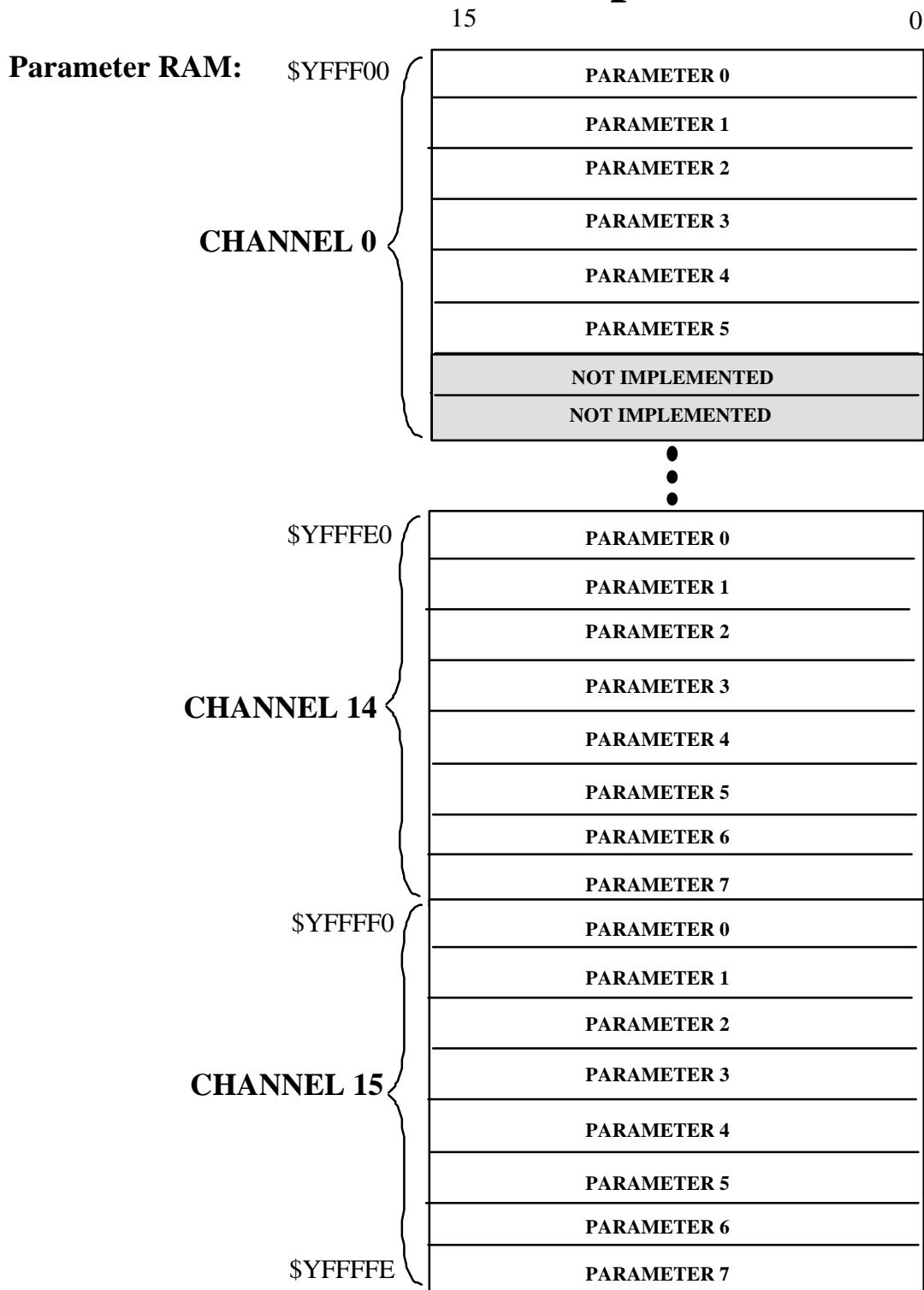
RAM

RAM Accesses



- **RAM subcommand allows TPU to access parameter RAM.**
- **Only P and DIOB can access RAM.**
- **All RAM accesses are WORD (16 bit) accesses.**
- **CPU and TPU both access RAM. For coherent 32 bit accesses, TPU must execute two consecutive RAM instructions, CPU must execute one 32 bit (longword) instruction.**

Parameter RAM Map



CHANNELS 0-13 have 6 parameters
CHANNELS 14-15 have 8 parameters.

RAM Subcommand I.

- **THREE forms of RAM subcommand:**

1) Channel Relative

Number in CHAN_REG determines the set of parameters (prm0-prm(5 or 7)).

EX. (*If CHAN_REG = 8. *)

```
ram    p -> prm0.      (* p is stored into Ch. 8's parameter 0 *)
```

2) Direct (Absolute)

User specifies Channel number and Parameter number.

EX. ram diob <- (14,7). (* Ch. 14's parameter 7 is loaded into diob *)
or ram diob <- \$EE. (*Each prm is word size, so the least significant nibble of address is 2 x prm number*)

3) by_diob (Indirect)

The address of the parameter is taken from diob 7:1. (Bit 0 is not used because all parameters are on word boundaries where bit 0 is 0.)

```
EX. au    diob := # $FC. (*load address to diob - Ch. 15 prm 6*)  
ram    p -> by_diob. (*p is stored at Ch.15 prm 6*)
```

NOTE: diob must be loaded with the RAM address **before** the instruction using by_diob.

- **Using Macros:**

EX.

```
%macro PERIOD_PWM 'prm1'
```

```
ram    p <- @PERIOD_PWM. (*prm1 loaded into p *)
```

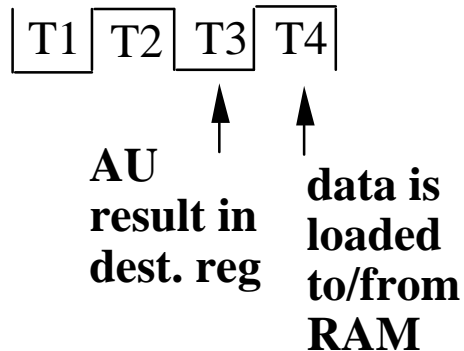
- **Functions using Channel Relative RAM accesses may be put on any channel.**

- **Absolute RAM accesses typically are used to access global parameters.**

- **Entry point RAM accesses must be Channel Relative.**

TEMPUS FUGIT

RAM Timing



- **REMEMBER THIS TIMING IN RELATION TO AU TIMING.**

- **AU result is loaded in destination in T3.**

*** Can do an arithmetic operation and move it to RAM in one instruction.**

- **CANNOT fetch a RAM parameter and use it in an arithmetic operation in one instruction.**

CORRECT EG.

```
au  p := p + sr;    (*result in p in T3 *)
ram p -> prm2.    (*store result in RAM in T4 *)
```

INCORRECT EG. (*Trying to fetch prm3 from ram and add it to ert in one instruction *)

```
ram p <- prm3;    (*fetches in T4*)
au  ert:= ert + p. (*result in ert in T3*)
```

Note: the order of the subcommands does not change the timing!

- **ALWAYS WRITE THE AU SUBCOMMAND BEFORE THE RAM SUBCOMMAND FOR TIMING CLARITY.**

Coherency

32 BIT COHERENCY

- **32 bit parameters must be read and written to coherently by the TPU and CPU.** This prevents a situation such as: the TPU updating one word of the parameter and before it updates the other word, the CPU reads the 32 bit parameter.
- **For the TPU: two consecutive RAM instructions guarantee that no CPU access will be allowed between.**
- **For the CPU: a longword read or write guarantees that no TPU access will be allowed before the entire 32 bits are accessed.**

EX. TPU:

```
ram    p -> prm2.      (*No CPU access allowed between*)
ram    diob-> prm3.    (*these 2 instructions*)
```

```
ram    p -> prm2.      (*Non-coherent writes to prm2*)
au     p := ert.      (*and prm3*)
ram    p -> prm3.
```

EX. CPU32:

```
move.l D2, prm2/prm3  (*Coherent write to prm2 & prm3*)
move.w #$1000, prm2   (*Non-coherent write to prm2 &*)
move.w #$5300, prm3   (*prm3*)
```

EX. CPU16:

```
sted   prm2/prm3      (*Coherent write to prm2 & prm3*)
ste     prm2           (*Non-coherent write to prm2 &*)
std     prm3           (*prm3*)
```

RAM Exercises

1. Give the subcommand to load parameter 2 associated with the current channel into p.
2. Give the subcommand to store the contents of diob to parameter 6 of channel 15.
3. Channel parameter 4, "TIMEADDR_ATM", (bits 7..1) contains the **address** of a parameter called "TIME_ATM". Write the instructions to load TIME_ATM into register p.
4. Write subcommands to coherently store a 32 bit value to the parameter ram.
5. What does the following sequence accomplish?

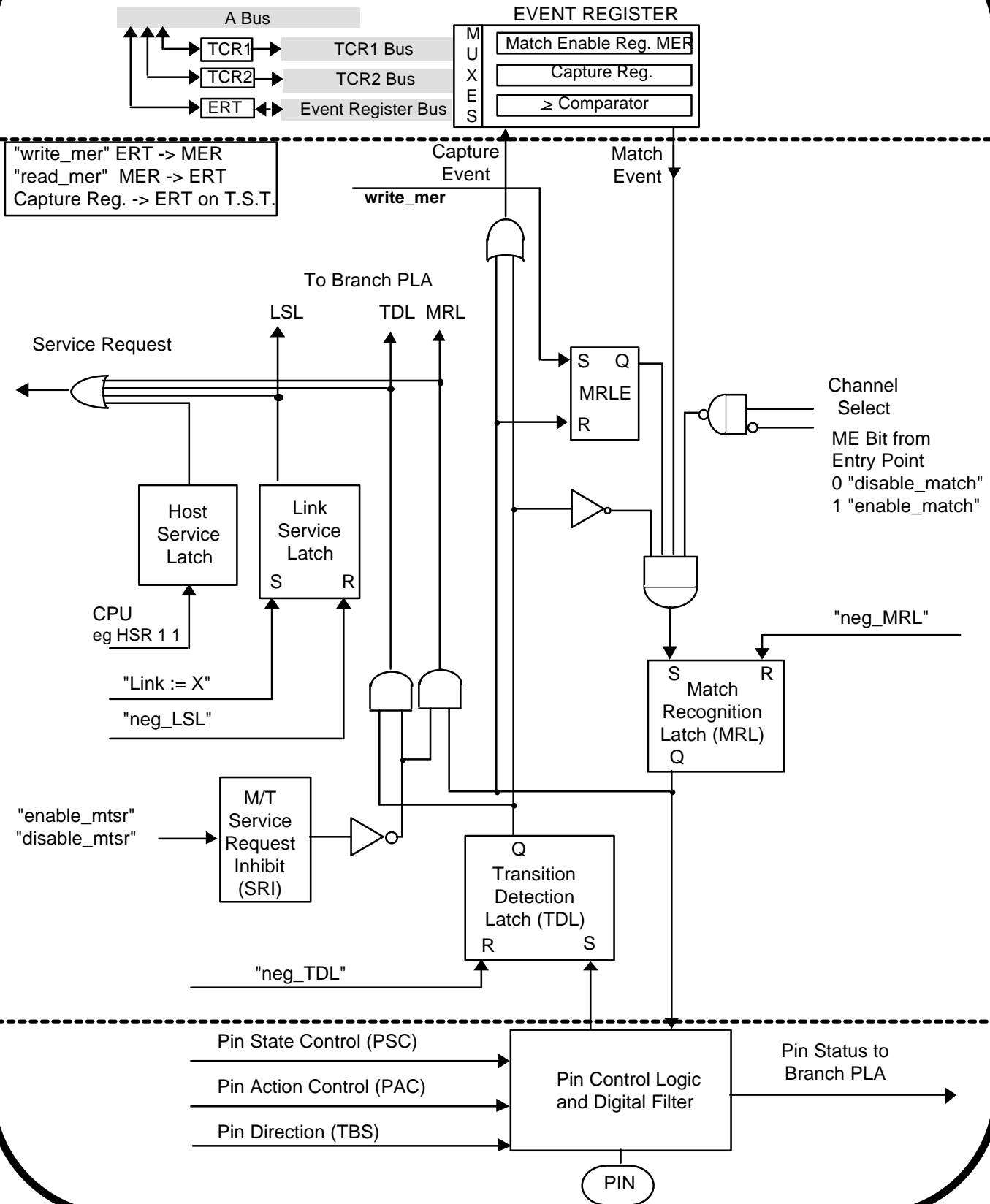
```
ram p <- prm1.  
ram p <- prm2;  
au a := p.  
au diob := a - p;  
ram diob -> prm1.
```
6. In exercise 5, how would you add an instruction to clear channel flag 1?

CHANNEL CONTROL

TEMPUS FUGIT

Channel Control MTT39 Rev 3.4 5 - 1

Channel Control



TEMPUS FUGIT

RESET State Channel Control & Latches

Channel Controls and Latches	Reset State
Match Recognition Latch (MRL)	Negated
Match Recognition Latch Enable (MRLE)	Negated Inhibits Assertion of MRL
Transition Detect Latch (TDL)	Negated
Match/Transition Service Request Inhibit (SRI) Latch	Asserted Inhibits MRL & TDL Service Requests, (But Not Assertion of MRL or TDL Latch)
Match Time Base	Use TCR1
Capture Time Base	Use TCR1
Pin Action Control (PAC)	No Transition will be Detected
Pin Directionality	Input

Chan Subcommand I.

TBS - Time Base and Pin Direction

tbs := in_m1_c1	0000
tbs := in_m1_c2	0001
tbs := in_m2_c1	0010
tbs := in_m2_c2	0011
tbs := out_m1_c1	0100
tbs := out_m1_c2	0101
tbs := out_m2_c1	0110
tbs := out_m2_c2	0111
No Change	1xxx

PAC- Pin Action Control

Output

pac := no_change	000
pac := high	001
pac := low	010
pac := toggle	011

Input

pac := no_detect	000
pac := low_high	001
pac := high_low	010
pac := any_trans	011
Nil	1xx

PSC- Pin State Control

PIN := PAC	00
PIN := high	01
PIN := low	10
NIL	11

eg.

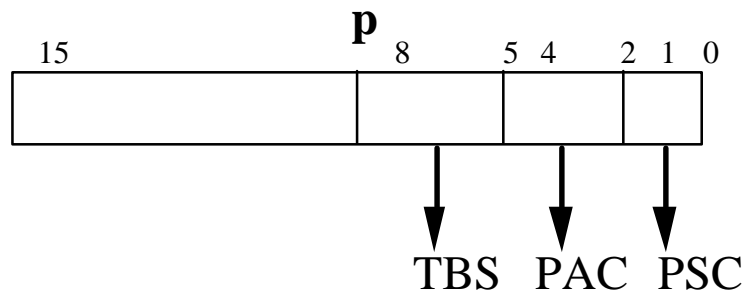
```
chan TBS := out_m1_c1,
      PIN := high,
      PAC := low.
```

(*output pin, match on tcr1, capture on tcr2; force pin high immediately; pin should go low on match*)

```
chan TBS := in_m1_c2,
      PAC := high_low
ram p <-- pram0.
```

(*input pin, match on tcr1, capture on tcr2, capture a high to low transition*)

config := p.



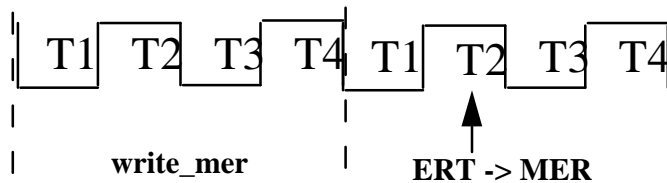
Chan Subcommand II.

Negate Latch(es)

chan neg_mrl, neg_tdl, neg_lsl.

Write Match Event Register (MER)

chan write_mer.



Enable/Disable Match and Transition Service Requests

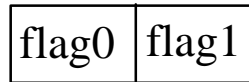
chan enable_mtsr.

chan disable_mtsr.

Chan Subcommand III.

SET/CLEAR FLAG0 or FLAG1.

```
chan set flag0
chan clear flag0
chan set flag1
chan clear flag1
```



TWO FLAGS per CHANNEL (internal to TPU)

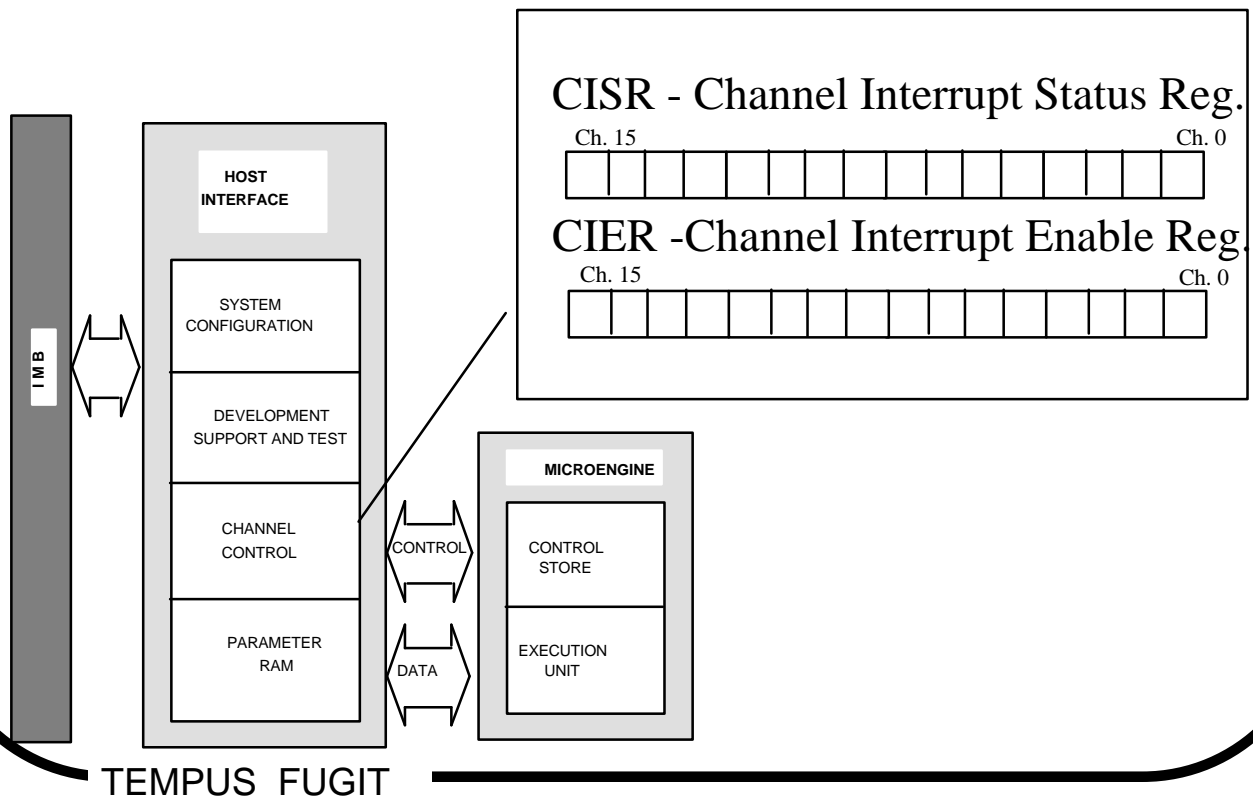
NOTES: Only ONE set/clear flag0/1 allowed per instruction.
flag0 is an entry point condition.

Send a CHANNEL INTERRUPT REQUEST to the CPU

```
chan cir.          ONE FLAG per CHANNEL (in CISR)
```

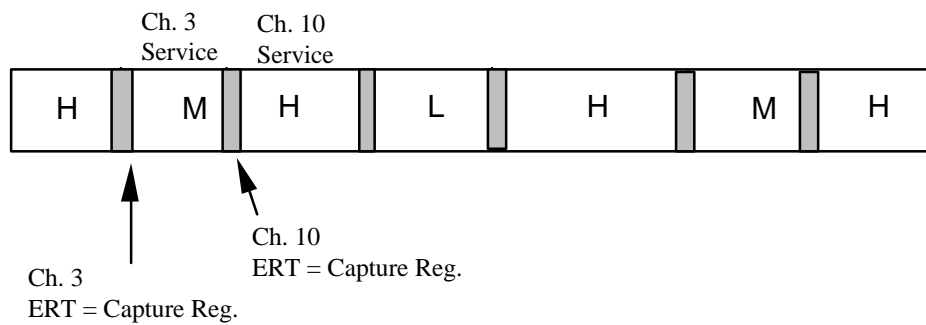
('cir' simply sets the channel's flag in CISR. If the channel's enable bit in CIER is set and TMCR and TICS are set up for interrupts then the channel will interrupt the CPU. The CPU could also poll CISR.)

in HOST INTERFACE



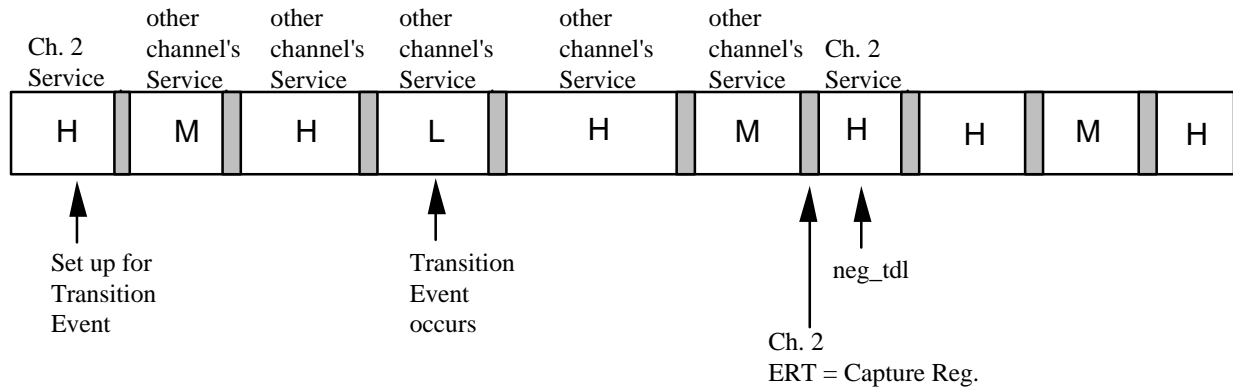
Capture Register to ERT

- **CAPTURE REGISTER** is loaded into ERT on **EVERY** Time Slot Transition



Transition Event

- **TRANSITION EVENT** causes a Capture.
- **TDL sets.**
- **TDL** being set blocks **MRL** from setting.
- **No other Capture** will occur until TDL is cleared.
- **If service is enabled** ("enable_mtsr"), TDL being set causes an m/tsr service request to be sent to Scheduler.



Match Event (1/2)

- **MATCH EVENT** causes a Match and a Capture.
- **Writing to MER** enables a Match (must write again to cause another Match).
- **MRL sets** after a Match. Must negate MRL before a new match.
- **If service is enabled** ("enable_mtsr"), MRL being set causes an m/tsr service request to be sent to Scheduler.
- **Match Recognition** can be **disabled** (i.e. MRL is blocked from setting) during the service of that channel ("disable_match" in entry point).

Match Even (2/2)

SET UP FOR A MATCH (example CH. 10)

```

ram    p <- @PARAMETERX.
chan   tbs := out_m1_c1,
       pin := low,
       pac := high,
       enable_mtsr.
au     ert := tcr1 + p;
chan   neg_mrl, neg_tdl, neg_lsl,
       write_mer;
end.

```

MATCH OCCURS (TCR1 \geq MER):

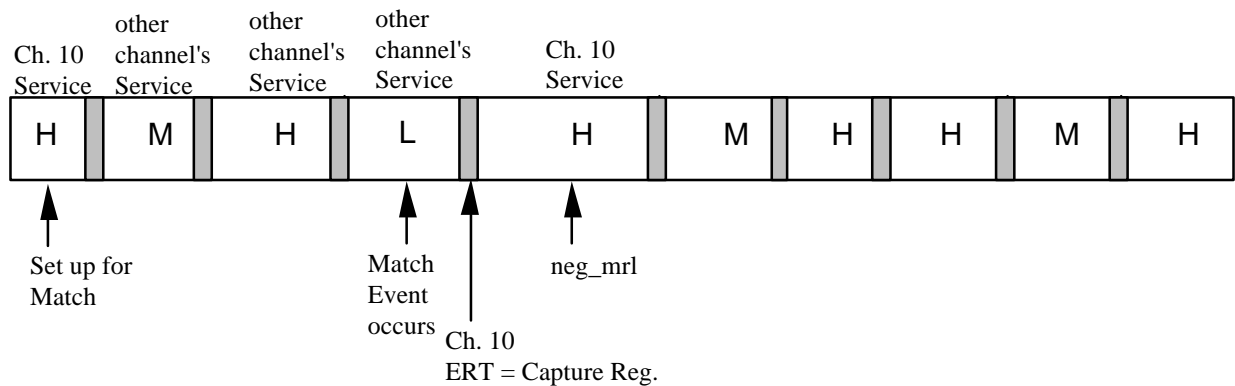
- MRL sets, pin goes high
- TCR1 -> Capture Register (TDL does not set)
- Ch. 10 requests m/tsr service

Next Service, NEGATE MRL to stop requests from that MATCH:

```

chan   neg_mrl.
(if desired set up for another match.)

```



Match Event and Transition Event

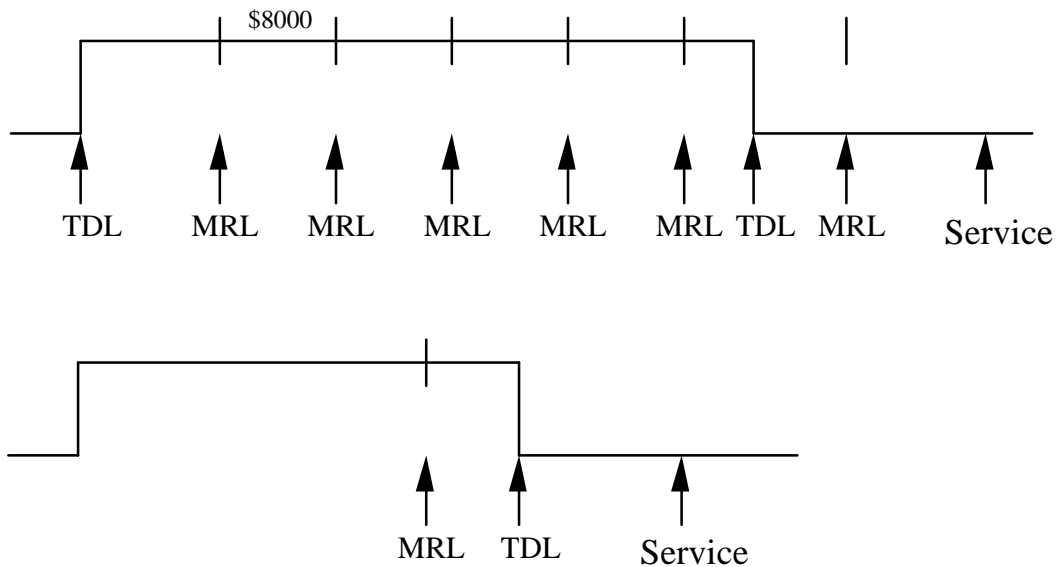
- TDL blocks MRL.
- MRL does not block TDL.

Why?

- M/TSR service request - 1 request for both msr and tsr.

EX.: If pin is input and using both transition and match:

Measuring a long pulse:



- If both MRL and TDL are set, MRL happened first.

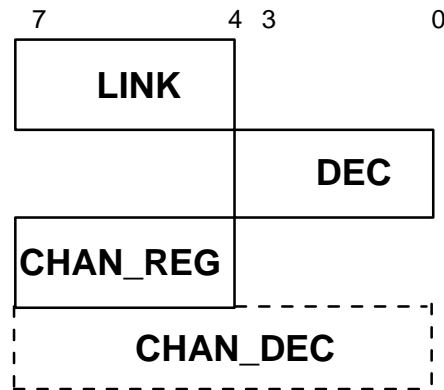
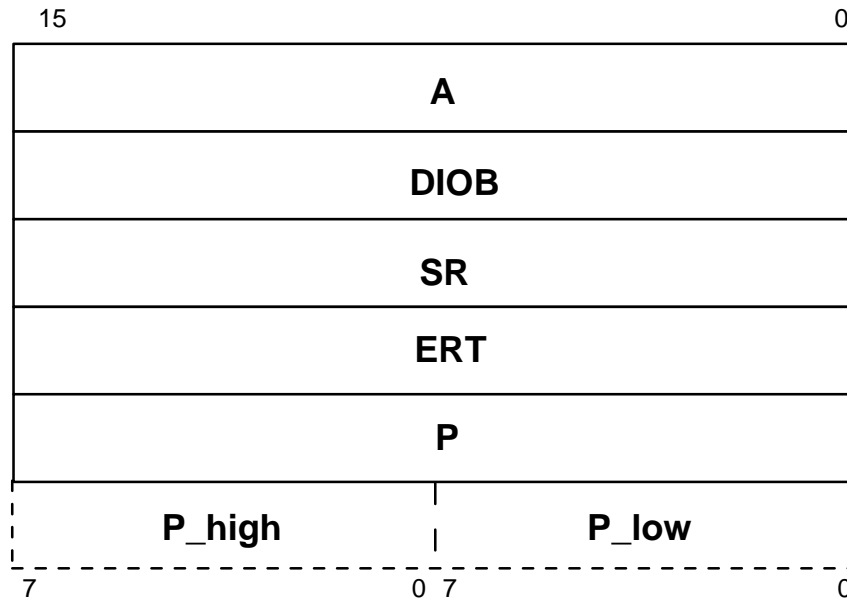
Channel Exercises

1. Give the subcommand to force the channel pin high immediately.
2. The channel pin is high and a match event is about to be scheduled. Give the subcommand to drive the pin low when the scheduled event occurs.
3. What subcommand(s) will initialize a channel to capture TCR1 and request service whenever any transition is detected on the pin?
4. Assume a state has been entered as a result of a link. Assume a match time has already been loaded into the ERT earlier in the state. Write the subcommands that will trigger a match without a pin transition (the pin is set up as an output), clear the appropriate latch(es), and request a host interrupt.
5. Register p is already loaded with configuration values. Write the instructions to configure using p, set flag1, clear flag0, and enable service requests.

EXECUTION UNIT

TEMPUS FUGIT

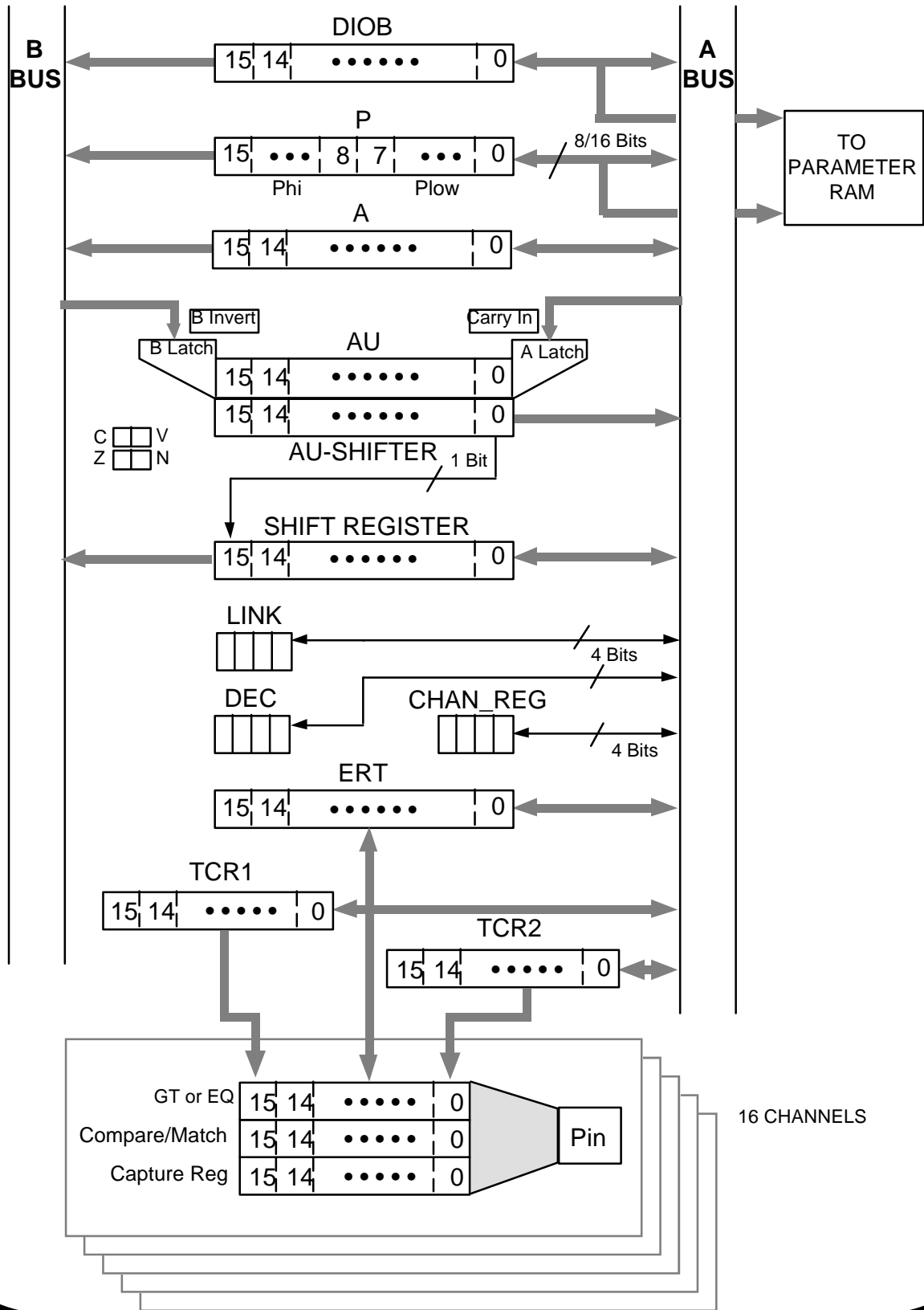
Register List



Name summary: A=Accumulator, DIOB= Data I/O Buffer, SR = Shift Register, ERT = Event Register Temporary, P = Preload Register, LINK= Link register, DEC=Decrementor Register, CHAN_REG=Channel Register

Note: All channels share these registers. Each channel has its own MER (Match Event Register) and Capture Register.

Execution Unit



TEMPUS FUGIT

AU Subcommand I.

INSTRUCTION FORMATS 1 and 2:

DESTINATION	OPERATOR	OPERANDS	0= On 1=Off BInv CIn	OPTIONS (for all cases in format 1. Not available in format 2.)	
destination	:=	0	1 1		
	:=<<	+1	1 0		
	:=>>	-1	0 1		
	:=R>	+max	0 0		
		asrc		1 1	shift
		asrc + 1		1 0	
		asrc - 1		0 1	ccl
		asrc + max		0 0	
		asrc + bsrc		1 1	
		asrc + bsrc + 1		1 0	
		asrc + !bsrc		0 1	
		asrc- bsrc - 1			
		asrc + !bsrc + 1		0 0	
		asrc - bsrc			

:=<< au shift left
:=>> au shift right
:=R> au rotate right
shift SR shift right (used for 32 bit shift)
ccl latch condition codes

destination

Word destination= a, diob, p, sr, tcr1, tcr2, ert, nil
 Byte destination = p_high(7:0), p_low(7:0), chan_dec(7:0)
 Nibble destination = link(7:4), chan_reg(7:4), dec(3:0)

asrc (a bus source)

Word source= p, a, diob, sr, tcr1, tcr2, ert, 0
 Byte source = p_high(7:0), p_low(7:0), chan_reg(3:0), dec(3:0), #0

bsrc (b bus source)

Word source= p, a, sr, diob, 0

special

read_mer (ert := mer on ER bus (T2); \$0000-> a bus)

AU Subcommand II.

INSTRUCTION FORMAT 5:

DESTINATION	OPERATOR	OPERANDS	OPTIONS (for all cases)
destination	:=	asrc	shift
	:=<<	asrc + immediate byte	
	:=>>	immediate byte	ccl
	:=R>		

:=<< au shift left
:=>> au shift right
:=R> au rotate right
shift SR shift right (used for 32 bit shift)
ccl latch condition codes

destination

Word destination= a, diob, p, sr, tcr1, tcr2, ert, nil
 Byte destination = p_high(7:0), p_low(7:0), chan_dec(7:0)
 Nibble destination = link(7:4), chan_reg(7:4), dec(3:0)

asrc (a bus source)

Word source= a, diob, p, sr, tcr1, tcr2, ert, 0
 Byte source = p_high(7:0), p_low(7:0), chan_reg(3:0),dec (3:0),#0

bsrc (b bus source)

#\$00-#\$FF on bits 7-0

special

read_mer (ert := mer on ER bus (T2); \$0000 -> a bus)

AU Subcommand Examples

Instruction Formats 1 and 2:

Note: "shift" and "ccl" are only available in format 1.

```
au    tcr1 := 0.
au    a := 1.
au    p_high := -1.
au    ert := max.          (*max = $8000*)
au    diob := tcr1, ccl.  (*latch cond. codes *)
au    p :=>> p + 1, ccl.  (*right shift result, latch cond. codes *)
au    nil :=<< p_low - 1, ccl. (*left shift result, latch cond. codes *)
au    a :=R> p + max.     (*rotate right result; max=$8000 *)
au    p := p + sr.
au    ert := p_low + a + 1, ccl.
au    a := diob + !p.     (*!p = one's compl. of p *)
au    a := diob - p - 1.  (*same instruction as above*)
au    diob := p + !diob + 1.
au    diob := p - diob.   (*same instruction as above*)
au    p :=>> p, shift.    (*32 bit shift of p and sr *)
au    read_mer.          (*ert:= mer *)
au    read_mer, p := diob. (*read_mer uses ER bus and a bus only*)
```

Instruction Formats 1 and 2

Illegal instructions examples:

```
au    a := p - max.      (*cannot subtract max, only add*)
au    ert:= p + tcr1.    (*tcr1 must be on A bus *)
au    a := tcr1 + tcr2.  (*tcr2 must be on A bus *)
au    p :=diob, read_mer. (*assembler puts diob on A bus *)
au    p := p_high + p_low. (*B bus source is word only *)
```


AU Subcommand Examples

Instruction Format 5

Legal Instruction examples:

```
au    ert := #40.          (*ert = decimal 40 *)
au    p_low := #$FF.      (*p_low := $FF *)
au    p_high := #$FF.     (*p_high := $FF, gets from b7-0 *)
au    a :=<< p + #3, ccl. (* result is left shifted, cond. codes latched. *)
au    ert :=>> p, shift.  (*32 bit shift, p and sr *)
au    read_mer.           (* ert := mer on ER bus *)
au    read_mer, p := #$80. (* ert := mer, p loaded with $0080 *)
au    chan_reg := #$90.   (*chan_reg := 9, gets from b7-4 *)
au    link := #$50.       (* link := 5, gets from b7-4 *)
```

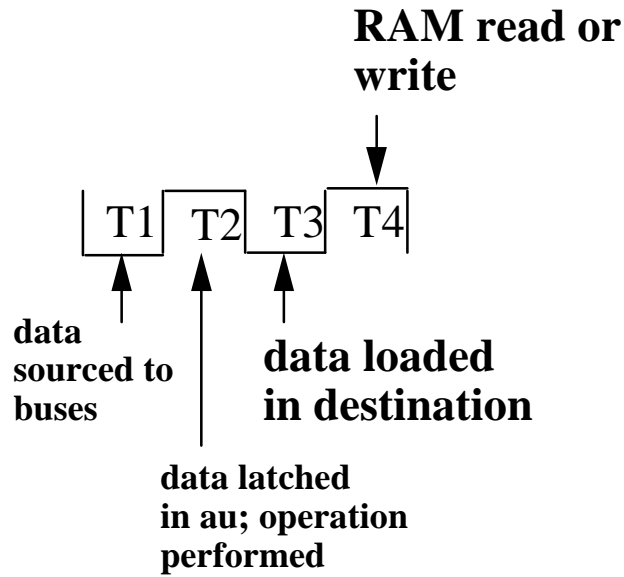
Instruction Format 5

Illegal instruction examples:

```
au    ert := #-2.         (*cannot use a "-" *)
au    p_low := #-1.       (*cannot use a "-" ; must say p_low := #$FF*)
```

NOTE: The symbol "#" is Motorola syntax used to denote immediate data. It is not necessary, but it may serve as a reminder that format 5 uses immediate byte data. Using a "#" sign with the internal constants 0, 1, -1, and max will force format 5 (0 and 1 can be used in formats 1, 2, and 5; -1 and max only in formats 1 and 2.)

AU Subcommand Timing



also for read_mer:
mer in ert during T2

Condition Codes

- **AU always adds words. Any source smaller than a word is padded with zeros. The result is either written to a word register; or part of the result is written to a byte or a nibble register; or the result is not written anywhere ("nil" destination).**

- **4 Condition Flags: C, V, Z, N are only latched when "ccl" is specified.**

- **Condition Codes are latched according to WORD result or BYTE result. Condition Codes are latched according to WORD result UNLESS:**

- 1) p_low or p_high is the destination

- 2) the sources are a byte of a register and an immediate value

EG. au diob := p_low + #2.

au a := chan_reg + #\$10. (all registers 8 bits or less are)

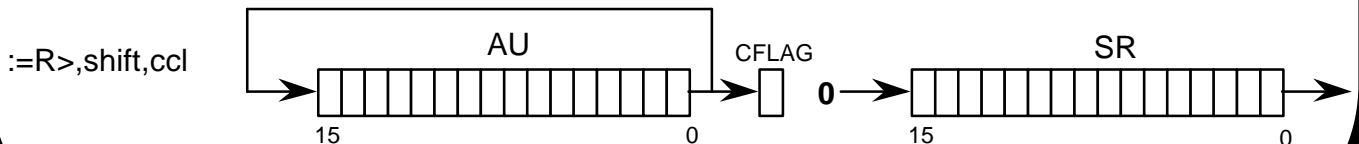
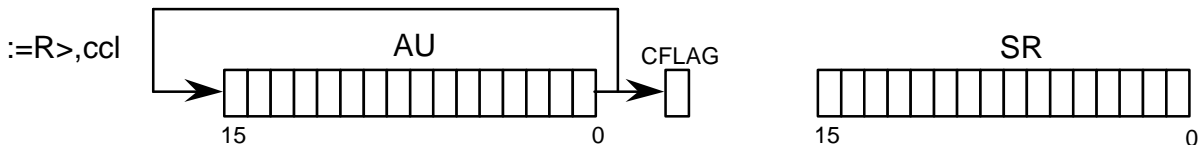
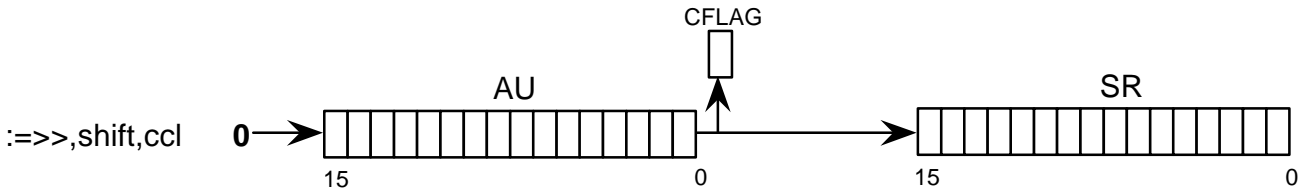
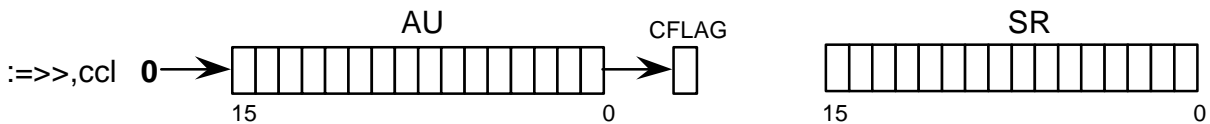
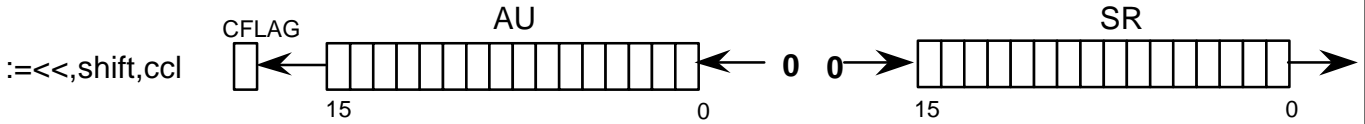
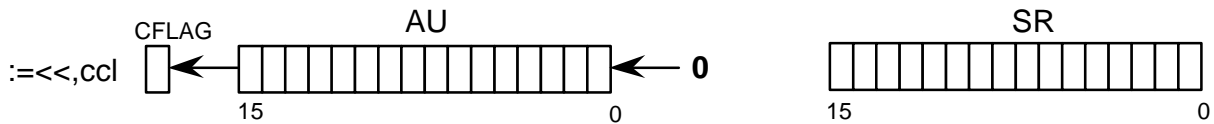
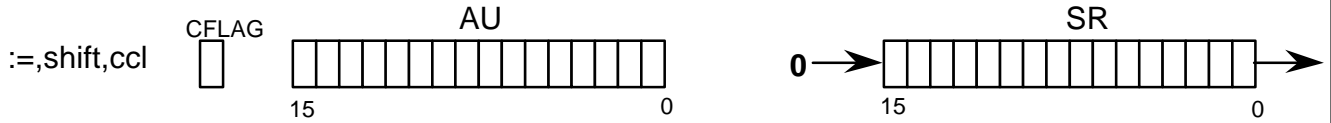
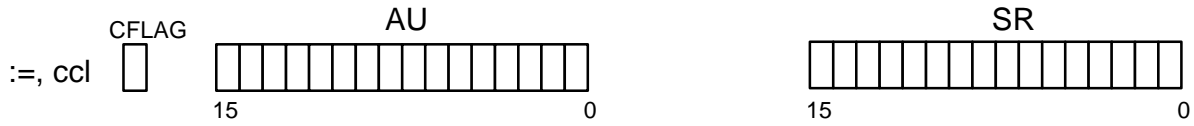
au dec := dec + #\$FF. (considered byte sources)

NOTE: If the AU shifter is shifting, the C flag is calculated on the word operation.

- **V always reflects the AU result before shifting.**

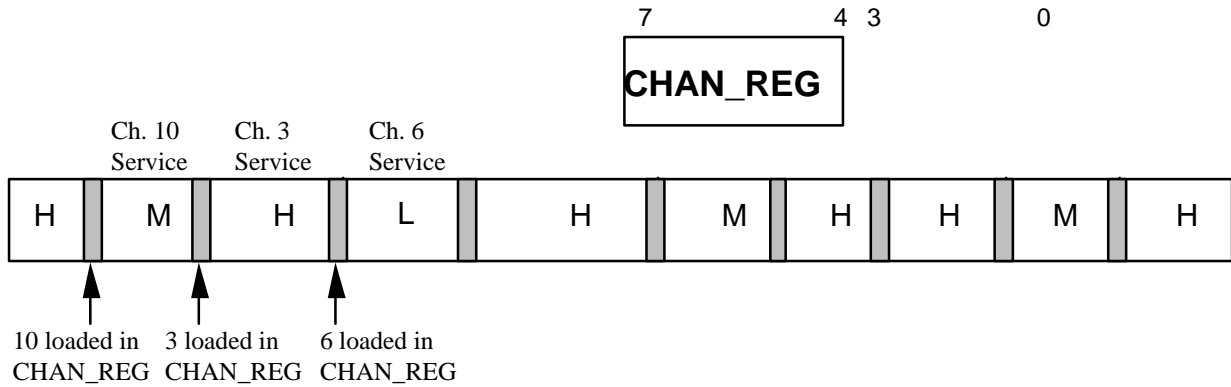
- **C, Z and N always reflect the output of the AU shifter which may or may not have shifted the AU result.**

AU Shifter and SR Shifter



TEMPUS FUGIT

Chan_Reg



- **CHAN_REG is loaded with bits 7:4**
- **CHAN_REG updated every Time Slot Transition to number of channel being serviced.**
- **Used to see environment of another channel**
- **Check timing chart to see when operations are available for new channel:**

Feature Used	Microcycle n	n+1	n+2	n+3
Write Channel Register	CHAN:= xxxx	New	New	New
RAM Commands Using CHAN	Old	New	New	New
Branch Using Pin State or Channel Flags	Old	Old	New	New
Branch on All Other Conditions (never accessible for new channel)	Old	Old	Old	Old
ERT Value (Capture Register -> ERT)	Old	Old	Old	New
Channel Commands: Neg_MRL Neg_TDL, TBS, PAC, PSC	Old	New	New	New
Channel Command Write MER	Old	Do Not Write	New	New
Channel Command Read MER	Old	Old	Do Not Read	New
Channel Commands Set/Clear Chan Flags enable_mtsr,disable_mtsr	Old	Old	New	New
Negate Link, CIR (never accessible for new channel)	Old	Old	Old	Old

Chan_Reg Examples

Check pin level on another channel:

```
au  chan_reg := #80.          (*change chan_reg to 8*)
nop.                          (*branch PLA uses pin state on old channel*)
if pin = 1 then goto PIN_HIGH. (*branch PLA uses pin state on new channel*)
.
.
.
```

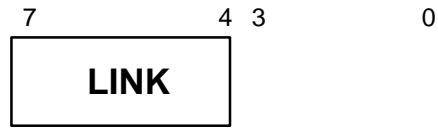
Get Capture register of new channel:

```
au  chan_reg := #F0.          (*change chan_reg to F*)
nop.                          (*ert unchanged*)
nop.                          (*ert unchanged*)
au  ert := ert - p.           (*ert=capture register of new channel*)
.
.
.
```

Change PSC on new channel:

```
au  chan_reg := #30.          (*change chan_reg to 3*)
chan PIN:= high.              (*force pin high on channel 3*)
.
.
.
```

Link

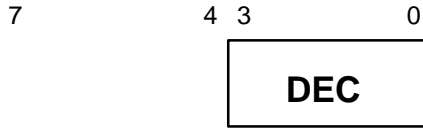


- **LINK := X** causes the Channel X's lsl to set, which causes an lsl service request to be sent to the Scheduler.
- **LINK** is load with bits 7:4

Examples:

au link := #\$70. (* causes Channel 7's lsl to set *)
au link := #\$F0. (* causes Channel F's lsl to set *)

Dec



DEC

- DEC loaded with \$F on every Time Slot Transition.
- 1 to \$F represent decimal values 1 to 15, 0 represents 16.
- DEC used with REPEAT and DEC_RETURN subcommands.
- DEC decrements once per μ cycle when used with REPEAT and DEC_RETURN:

DEC with REPEAT:

Instruction with the repeat subcommand executes DEC+1 times, then DEC is reset to \$F.

Example:

```
au   dec:=#5.
repeat;
au   p:=>>p.      (* this instruction executes 6 times *)
au   ert := p+1.
```

DEC with DEC_RETURN:

Only part of subroutine is executed: the number of instructions executed in the subroutine = the number in DEC. After returning from subroutine, DEC is reset to \$F.

Example:

```
au   dec := #4.      (*execute 4 instructions in the subroutine*)
call SUB1; dec_return.
    .
    .
    .
```

SUB1:

```
au   p := p + 1.
chan set flag1.
au   link:= #80.
au   chan_reg:= #80. (* after this instruction returns to main program; DEC=$F*)
au   ert := max.
    .
    .
    .
return.
```


Multiply Routine

- The TPU can perform a 16x16 bit multiply with a 32 bit result.
- Multiplying takes the form of $\text{reg1:sr} = \text{reg2} * \text{sr}$ where before execution
 - reg1=0
 - reg2=first 16 bit operand
 - sr = second 16 bit operand
- p, a, diob, and ert can be used for reg1 and reg2

- **Multiply routine:**

(* reg1 contains 0 reg2 contains a 16 bit operand, sr contains a 16 bit operand*)

```
au dec := #15. (*repeat addition loop 16 times*)
repeat;
au reg1 :=>> reg1 + reg2, shift. (*16 shifts & adds = 16x16 bit multiply*)
```

MULTIPLY EXAMPLE:

(* diob = first operand, sr = second operand, p = 0 *)

```
au dec := #15. (*repeat addition loop 16 times*)
repeat;
au p :=>> p + diob, shift. (*16 shifts & adds = 16x16 bit multiply*)
```

(*result is now in p:sr *)

Multiply Routine Explanation

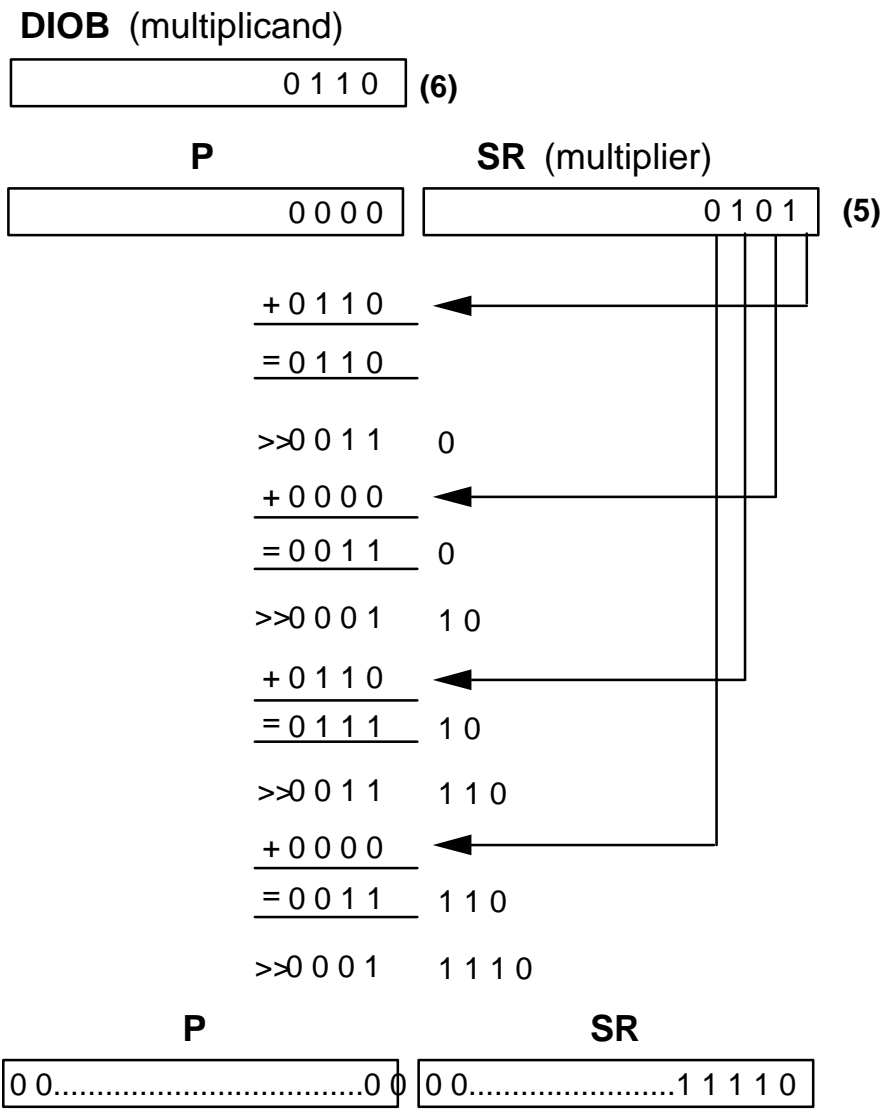
(* diob = first operand, sr = second operand, p = 0 *)

```

au dec := #15.          (*repeat addition loop 16 times*)
repeat;
au p :=>> p + diob, shift.  (*16 shifts & adds = 16x16 bit multiply*)
    
```

(*result is now in p:sr *)

- In the combined condition of: the SR is shifting (shift), the AU shifter is right shifting (:=>>) and DEC is decrementing (repeat), the least significant bit of sr will control the b bus input to the au. If sr = 1, the b bus input to the au is the contents of the b bus (in this example, diob is on the b bus). If sr = 0, the b bus input to the au is zero.



p:sr = 30 after 16 iterations

AU Exercises

1. Give the subcommand to add p to tcr1 and place the result in ert.
2. Give the subcommand to increment the low byte of p.
3. Show an example of a 32 bit right shift.
4. Write a subcommand that compares diob and tcr2 for later branching.
5. What is wrong with the following subcommand?

```
au a := a - #3;
```

6. What does the following sequence accomplish?

```
au sr := p.  
au a := 0.  
au dec := #15.  
repeat; au a :=>> a + p, shift.
```

MICROENGINE/ SEQUENCING

Sequencing Subcommands

- SEQUENCING SUBCOMMANDS control the μ PC:

goto (label){ flush }	(*jump - format 4*)
if (condition) then goto (label){ flush }	(*conditional branch -format 3*)
call (label){ flush } {; dec_return }	(*jsr -format 4*)
return { flush }	(*return from subroutine -format 4*)
repeat	(*repeat current instruction - formats 1,2,5*)
end	(*end state - formats 1,2,4,5*)

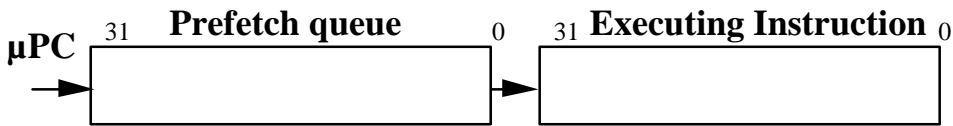
- Note:** labels must end with a colon (:).

Ex. goto LABEL_ATX.

•
•
•

LABEL_ATX:

Flush



- μPC prefetches the next in line instruction
- For change -of -flow subcommands:

```
goto
if ... then
call
return
```

you may choose to EXECUTE the prefetched instruction (No flush) or FLUSH the prefetched instruction (a one μcycle NOP) (Flush).

- NO FLUSH is DEFAULT.

EX.

```

    goto HADES.                (*may also say "goto HADES, no_flush"*)
    au   p := p + 1.           (*this instruction is executed*)
    au   ert := max.
    .
    .
HADES: if N = 1 then goto XINGU, flush.
    ram  p -> prm3.           (*this instruction is not executed*)
    au   ert := ert + p.
    .
    .
XINGU: call DANUB, flush.
    au   p_low := p_low + #2, ccl. (*this instruction is not executed until*)
    au   diob := 0.          (*return from subroutine*)
DANUB: .
    .
    return.
    ram  p -> prm4.           (*this instruction is executed*)
```

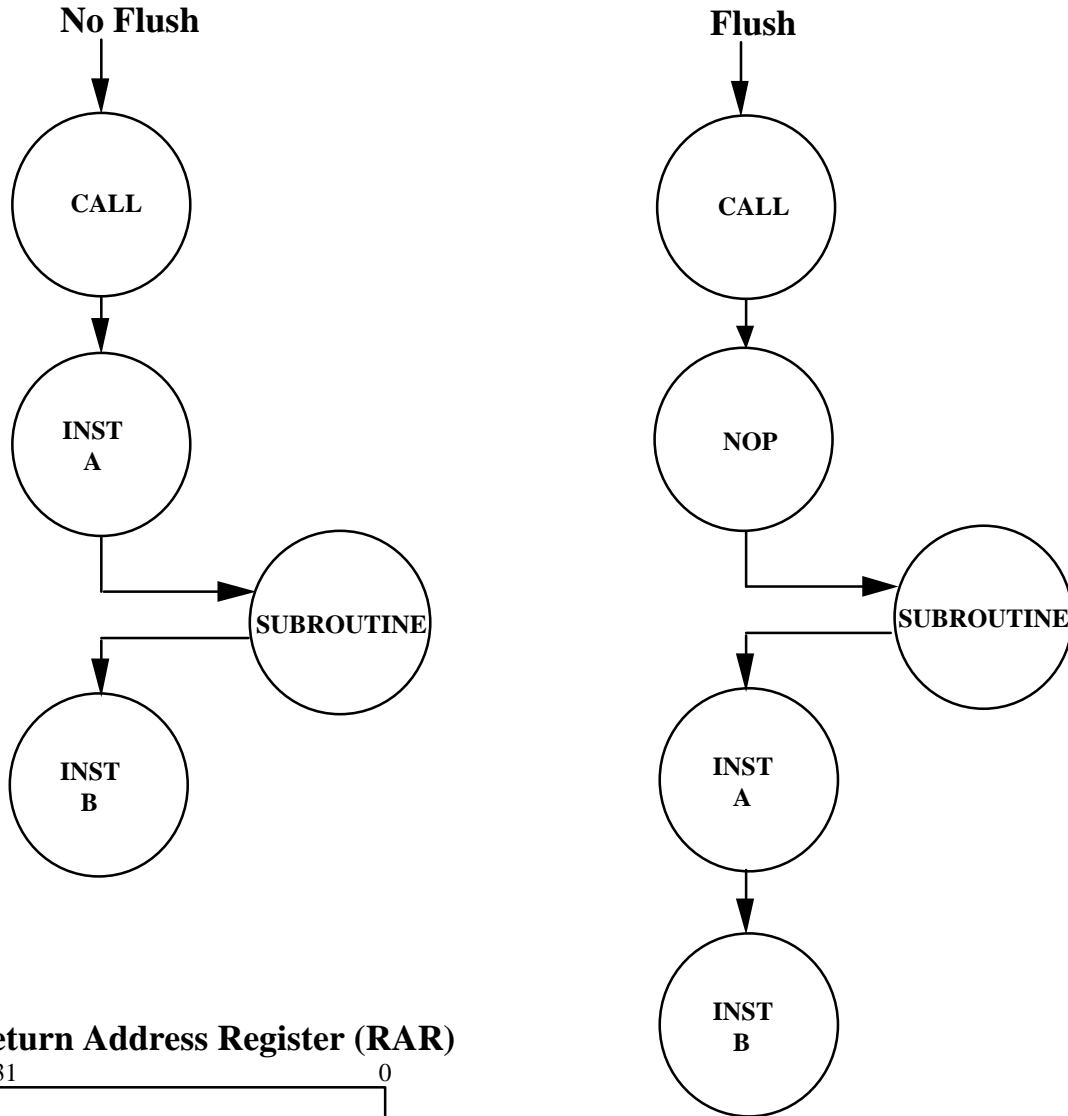
Flush with Subroutine Calls

call Subroutine, {flush}.

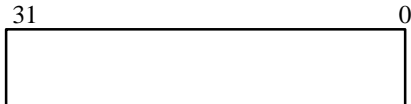
instr. A

instr. B

•
•
•



Return Address Register (RAR)



NOTE: The Return Address for subroutines is stored in the RAR. There is only one RAR; therefore only one level of subroutines is valid (i.e. no nested subroutines).

Branch PLA Flags

For Condition Branches

if (condition) **then goto** (label){,flush}

These conditions may be used:

Name	Description
V	Overflow from AU; byte or word;
N	Negative from Shifter; byte or word;
C	Carry from AU; byte or word;
Z	Zero from Shifter; byte or word;
LOW_SAME	(C + Z) asrc is lower/same as bsrc (unsigned branch)
LESS_THAN	$N \cdot !V + !N \cdot V$ asrc is less than bsrc (signed branch)
FLAG1	Channel Flag1; one for each channel;
FLAG0	Channel Flag0; one for each channel;
TDL	Transition Detection Latch;
MRL	Match Recognition Latch;
LSL	Link Service Latch;
HSQ1	Sequence bit 1; one for each channel;
HSQ0	Sequence bit 0; one for each channel;
PSL	Pin State Latch

NOTES:

- MRL, TDL, LSL & HSQ1/0 are latched into the PLA during each Time Slot Transition.
- Pin status is latched into the PLA during Time Slot Transition AND when CHAN_REG is changed via microcode.
- Z,N,V,and C are only latched when the microcode specifies (ccl).

IF... THEN Examples

EX.

```
au  chan_reg := chan_reg.          (* relatch pin state *)
nop.                               (*wait for pin state to re-latch *)
if PSL = 1 then goto LABEL0.      (*checking pin state condition *)
ram <- @PAR_SQW.                  (*this instruction is executed*)
```

•
•
•

```
au  nil:= sr - diob, ccl.          (*compare sr and diob as signed #s*)
if LESS_THAN=TRUE then goto LABEL1. (*if sr is less then diob goto label1*)
```

•
•
•

```
au  sr := sr - diob, ccl.          (*compare sr and diob as unsigned #s*)
if LOW_SAME=FALSE then goto LABEL2. (*if sr is higher than diob goto label2*)
```

•
•
•

```
if HSQ0 = TRUE then goto LABEL3, flush. (*flush the next instruction*)
```

•
•
•

```
if HSQ1 = FALSE then goto LABEL4, flush. (*0 is alternative syntax to FALSE*)
if HSQ0= TRUE then goto LABEL5.
```

(*Note: if conditional branches are to be written back to back as in the above example, a "flush" in the first instruction is strongly recommended.*)

•
•
•

```
if HSQ1 = 1 then goto LABEL6.      (*1 is alternative syntax to TRUE*)
```

•
•
•

```
if TRUE then goto LABEL7.          (*This forces format 3. This may be*)
                                   (*useful if you need some other*)
                                   (*format 3 fields with a non-conditional*)
                                   (*jump*)
```

REPEAT and DEC_RETURN (with DEC)

DEC

- DEC loaded with \$F on every Time Slot Transition.
- 1 to \$F represent decimal values 1 to 15, 0 represents 16.
- DEC used with REPEAT and DEC_RETURN subcommands.
- DEC decrements once per μ cycle when used with REPEAT and DEC_RETURN:

DEC with REPEAT:

Instruction with the repeat subcommand executes DEC+1 times, then DEC is reset to \$F.
EX.

```
au    dec:=#5.
repeat;
au    p:=>>p.      (* this instruction executes 6 times *)
au    ert := p+1.
```

Note: REPEAT is only available with formats with AU subcommand.
I.e., formats 1, 2, and 5 but not format 4.

DEC with SUBEN:

Only part of subroutine is executed: the number of instructions executed in the subroutine = the number in DEC. After returning from subroutine, DEC is reset to \$F.
EX.

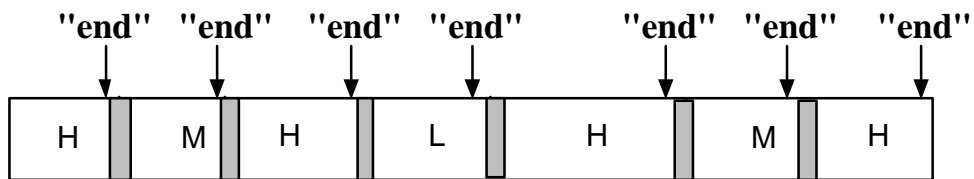
```
au    dec := #4.      (*execute 4 instructions in the subroutine. This is*)
call SUB1; dec_return. (*true for both flush and no flush. *)
    .
    .
    .
```

SUB1:

```
au    p:=p+1.
chan  set flag1.
au    link:=#$80.
au    chan_reg:=#$80. (* after this instruction returns to main program; DEC=$F*)
au    ert := max.
    .
    .
    .
return.
```

END

- "end" ends the state.
- Scheduler then schedules next channel.



NOTE: After becoming familiar with TPU microcode, refer to the TPU Programmer's Reference Manual for the few restrictions on certain instruction combinations.

Sequencing Exercises (1/2)

1. Give a subcommand to unconditionally jump to a location identified by the label "The_Devil" (there are 2 correct answers).

2. Given the following au subcommand: `au nil := diob - p, ccl.`

Diob and p contain unsigned numbers. Write the subcommands to execute at the label "Frying_Pan" if diob is greater than p; execute at the label "Fire" if diob is less than p; and execute at the label "Home_Safe" if diob is equal to p.

3. A state in a function involves more than one channel. What sequence would be required to make a branch decision based on the state of the pin associated with the next higher channel number?

Sequencing Exercises (2/2)

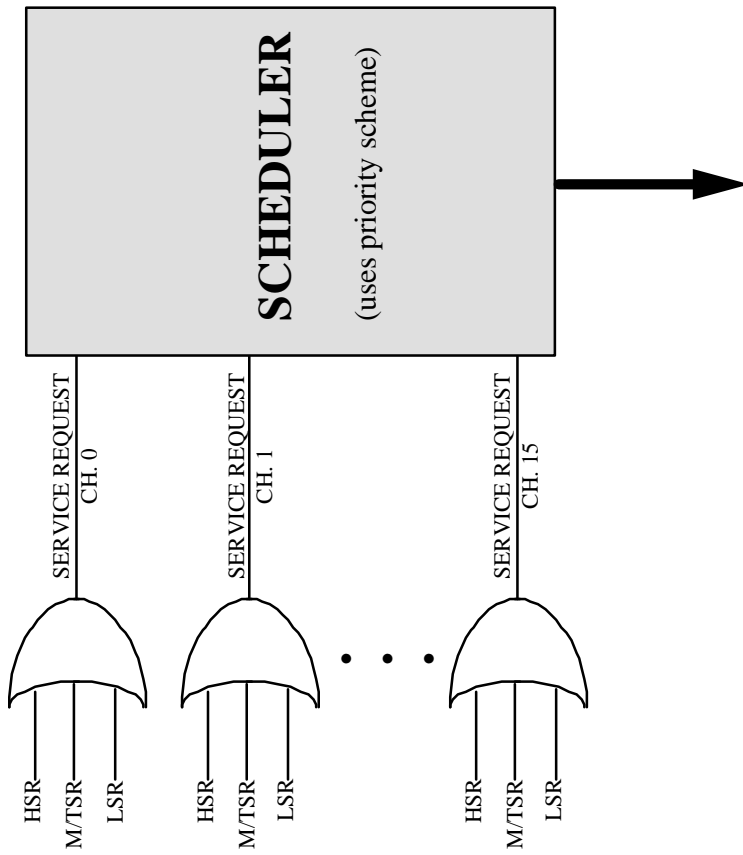
4. If a state of an input function is entered as a result of a m/tsr, what is the recommended conditional branch to vector the function to either the label "Match_Detected" or the label "Transition_Detected"?

5. What does the following sequence accomplish if a equals p? (Hint: circle each instruction, and don't forget about flushes and no flushes.)

```
au    nil := a - p, ccl;
chan  set flag1.
If Z = 1 then Goto Confusion;
chan  set flag0.
chan  clear flag1.
chan  clear flag0.
Confusion:
  If flag1 = 0 then Goto More_Confusion, flush.
Complete_Confusion:
  end.
More_Confusion:
  chan  cir;
  end.
```

SCHEDULER

SCHEDULER

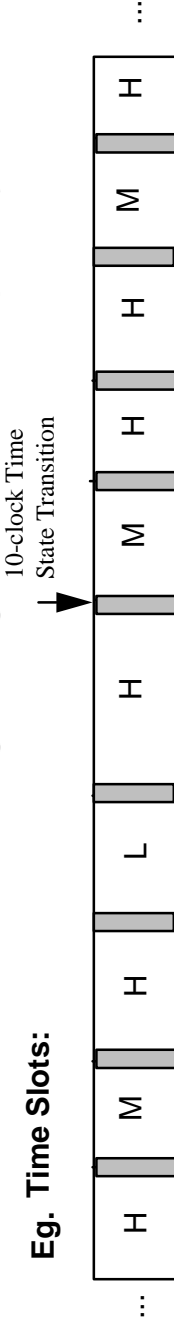


Channel to be serviced next

- HSR** - Host Service Request (from CPU)
- M/TSR** - Match or Transition Service Request (from Channel hardware)
- LSR** - Link Service Request (from Execution Unit)

PRIORITY SCHEME - Primary Level

- Each active channel must be assigned High, Middle, or Low priority



- Scheduler assigns time slots in repeating pattern: H • M • H • L • H • M • H
- Priority-passing occurs if no channel in a priority level is requesting service

ORDER OF PRIORITY-PASSING:

Assigned Priority Level		Next Priority Level		Next Priority Level
High	↑	Middle	↑	Low
Middle	↑	High	↑	Low
Low	↑	High	↑	Middle

NOTE: If the Scheduler has no new requests pending, it will remain at the current slot waiting for an event.

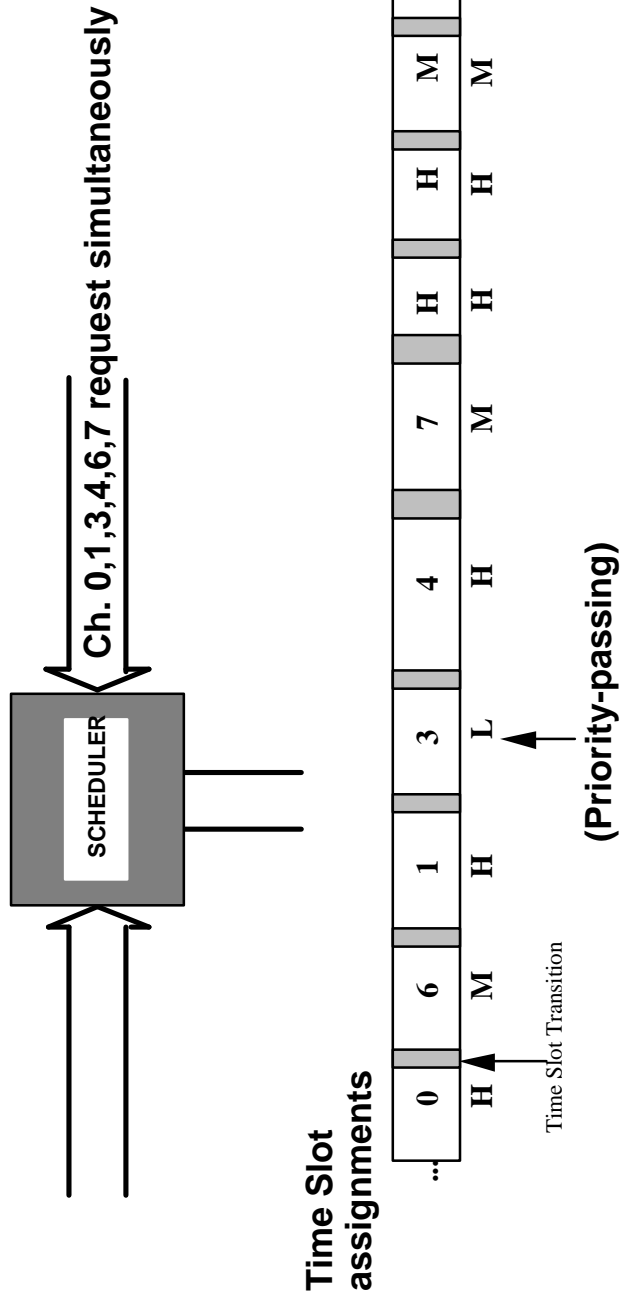
PRIORITY SCHEME - Secondary Level

- If Channels of the same priority simultaneously request service, the lowest numbered channel is granted service first.

e.g. Channels 0-4 High priority

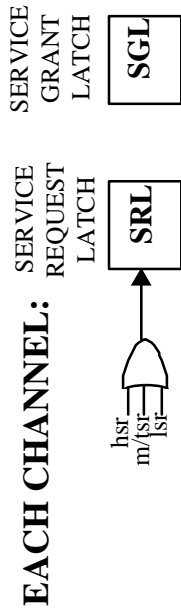
Channels 5-8 Middle priority











Channels 9-15 Low priority



ROUND ROBIN

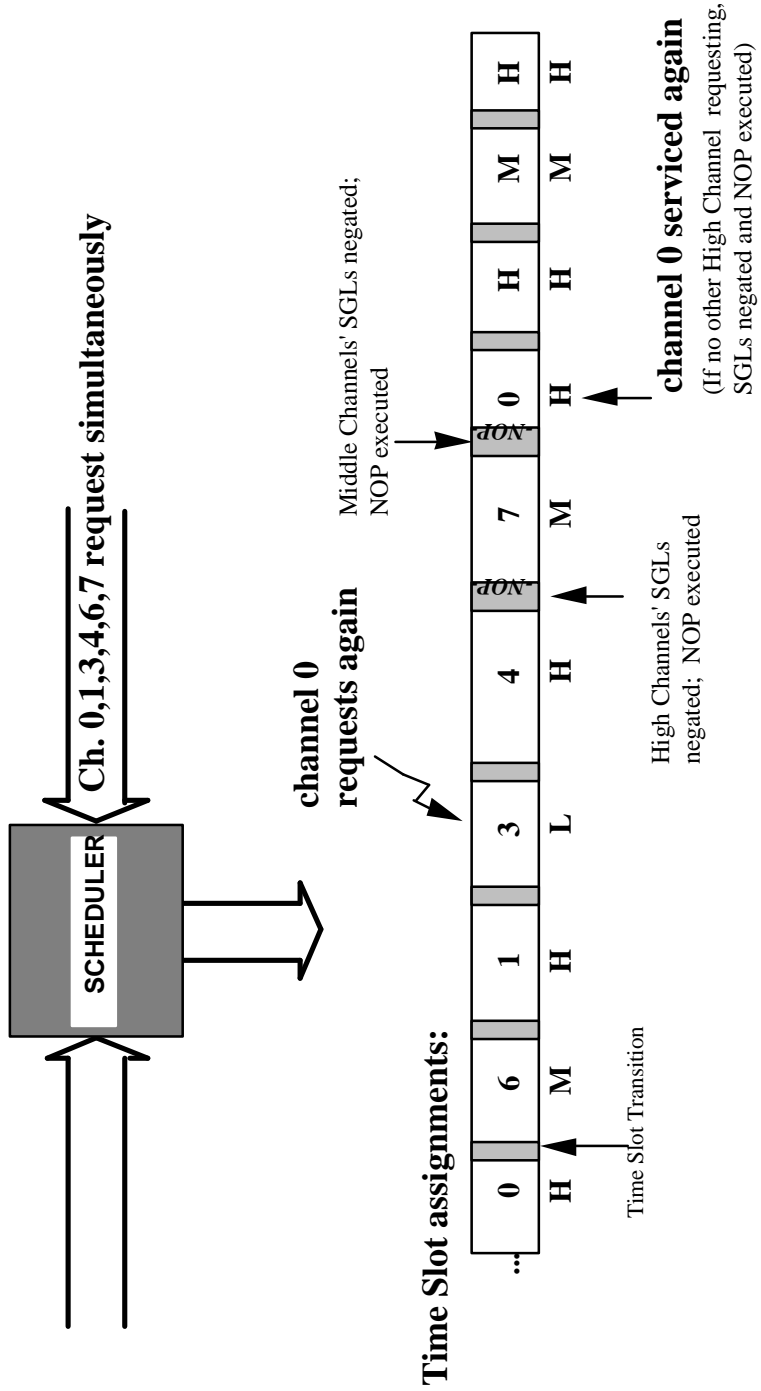
Each priority level operates **ROUND ROBIN**. This ensures that high-numbered channels as well as low-numbered channels are granted service. **EACH CHANNEL:**



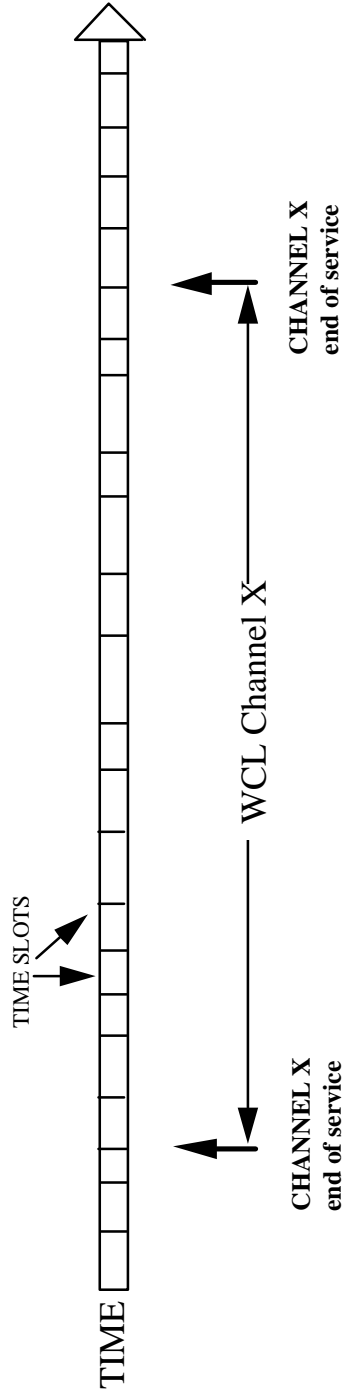
- When a channel requests service, its SRL asserts.  
- When a channel has been granted service, its SGL is asserted by Scheduler hardware and the latch that caused the request (hsl, mrl, tdl, or lsl) is negated (hsl by the hardware, mrl, tdl, lsl in microcode).  
- If the channel has another request an asserted SGL disables the SRL from being recognized. The new request will be recognized when SGL is negated.  
- SGLs are negated in priority groups: high channels' SGLs are negated together, middle channels' SGLs are negated together and low channels' SGLs are negated together.  
- A priority groups' SGLs are negated at the Time Slot Transition **after** a time slot of that priority level, **if for that priority level:**
 - No channel has an SRL asserted **and** an SGL negated:  
- When a priority level's SGLs are negated a 4 CPU clock NOP is executed

ROUND ROBIN Example

- e.g. Channels 0-4 High priority
- Channels 5-8 Middle priority
- Channels 9-15 Low priority



WORST CASE LATENCY



To calculate Worst Case Latency (WCL) for Channel X, assume that Channel X has just been serviced. Assume all channels are continuously requesting service. What is the worst case time until the end of Channel X's next service?

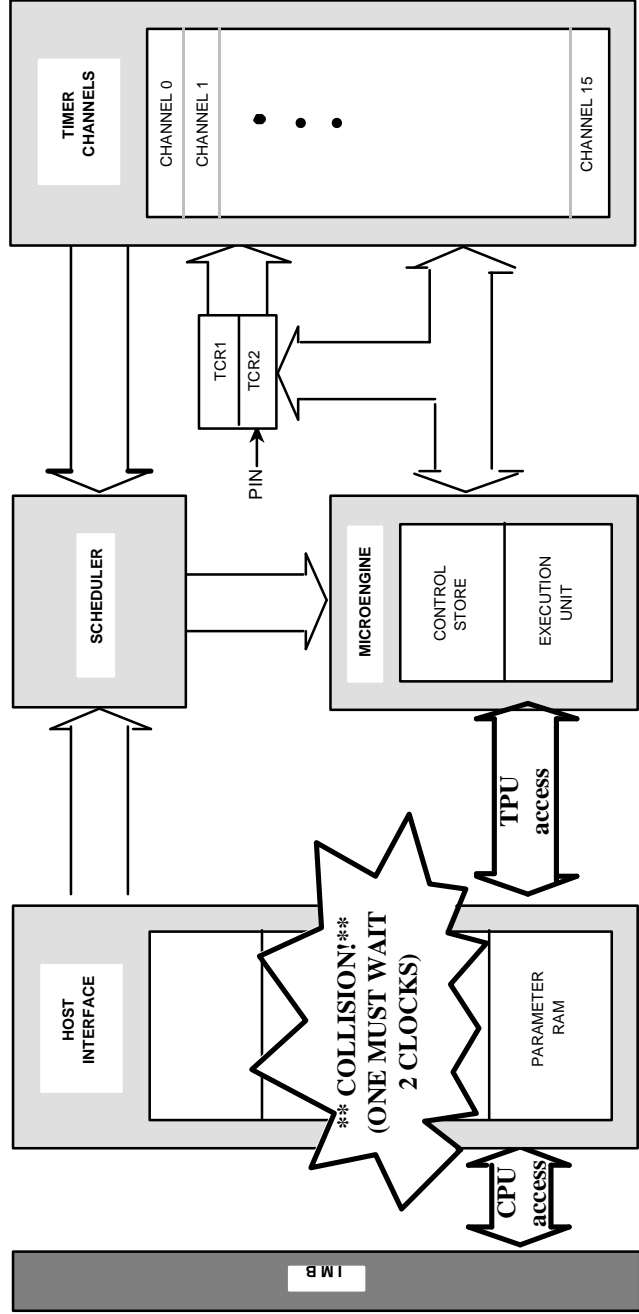
- 1) Find Worst Case Service Time for Each Channel
- 2) Using H-M-H-L-H-M-H schedule, map time slots granted until Channel X end of service.
- 3) Add time for Time Slot Transitions and NOPs executed.

WCL DEFINITIONS

Worst Case Service Time (WCST) = (longest state) + (number of TPU RAM accesses during state) * RCR * 2 clocks

where RCR = RAM collision rate, an approximation of the percentage of time the CPU and the TPU try to access the RAM simultaneously.

Note: If 2 RAM accesses are back-to-back, count them both as one RAM access because the second access will never have to wait for the CPU.



Time Slot Transition Time = 10 clocks (always).

NOP = 4 clocks, executed at end of a Round Robin for a priority level.

WORST CASE LATENCY EXAMPLE

Channels 0-3 HIGH priority
 Channels 4-7 MIDDLE priority
 Channels 8-15 LOW priority

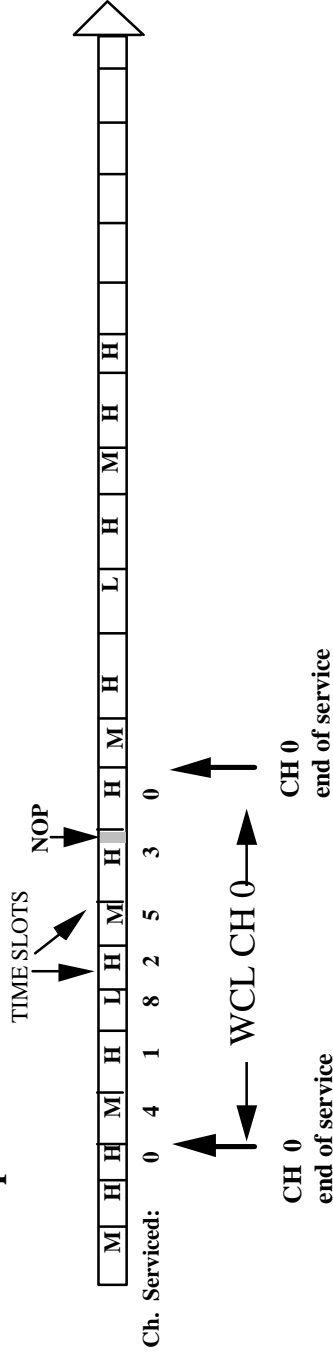
All Channels executing PWM
 RCR = .50

1) Worst Case Service Time for PWM = (longest state) + (number of TPU RAM accesses during state) * RCR * 2 clocks
 = 24 clocks + 4 * 1/2 * 2 clocks
 = 28 clocks

2) Map according to H-M-H-L-H-M-H

(A) MAP HIGH CHANNEL WORST CASE LATENCY:

Example Channel 0:



Assume Channel 0 has just been serviced and that all other channels are continually requesting service.

Because of Round Robin, all other High Channels will be granted Service before Channel 0.

2 Middle Channels granted. 4 * 28 clks = 112 clks

1 Low Channel granted. 2 * 28 clks = 56 clks

1 NOP for end of High Ch. Round Robin 1 * 28 clks = 28 clks

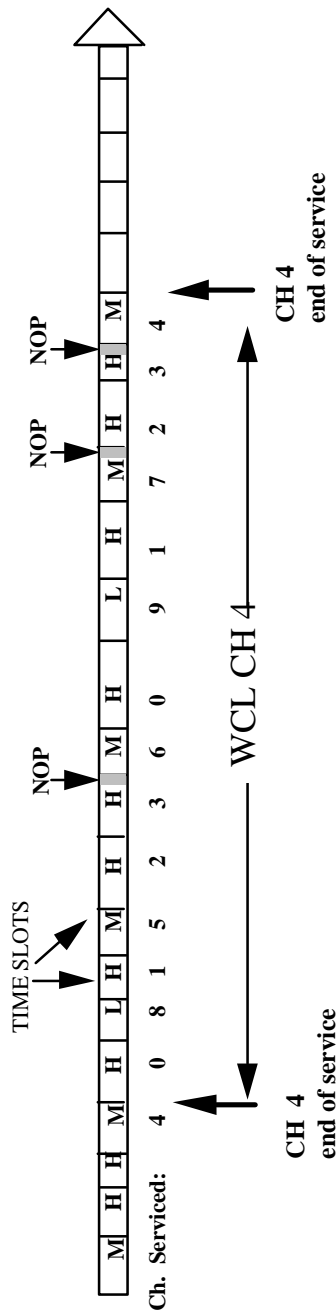
7 Time Slot Transition Times 4 clks

WORST CASE LATENCY CH. 0 70 clks

270 clks

WORST CASE LATENCY EXAMPLE cont.

B) MAP MIDDLE CHANNEL WORST CASE LATENCY: Example Channel 4:



4 Middle Channels granted
 8 High Channels granted.
 2 Low Channels granted.
 3 NOPs
 14 Time Slot Transition Times
WORST CASE LATENCY CH. 4

4 * 28 clks = 112 clks
 8 * 28 clks = 224 clks
 2 * 28 clks = 56 clks
 12 clks
 140 clks
544 clks

C) MAP LOW CHANNEL WORST CASE LATENCY: Example Channel 8:

8 Low Channels granted
 32 High Channels granted.
 16 Middle Channels granted.
 13 NOPs
 56 Time Slot Transition Times
WORST CASE LATENCY CH. 8

8 * 28 clks = 224 clks
 32 * 28 clks = 896 clks
 16 * 28 clks = 448 clks
 52 clks
 560 clks
2180 clks

SCHEDULER EXERCISES (1/2)

For all Scheduler Exercises, use a 16.67 MHz system clock, and a system clock $\div 4$ TCR1. Use Ram Collision Rate of .20 for question 1 and 0 for question 2.

1. Using the SQW function you wrote in Programming Lab 1, find the Worst Case Latency on a channel when 3 high priority channels are running SQW. Compare this to the measurements you took in Lab 1.

2. Assign SQW to Channel 0 as High priority, Channel 1 as Middle priority, and Channel 2 as Low priority. Find the Worst Case Latency for each channel.

SCHEDULER EXERCISES WORKSHEET

TEMPUS FUGIT

KNOWLEDGE CHECK

KNOWLEDGE CHECK

TPU MICROCODE PROGRAMMING

1. A TPU microcode instruction takes how many CPU system clock cycles to execute (not counting Ram Collision time)?
 - a) 1
 - b) 2
 - c) It depends on the prescale values for TCR1 and TCR2.
 - d) It depends on the instruction format.
 - e) None of the above
2. If a match event occurs for a given channel and before it is scheduled a Host Service Request 10 is made for the same channel, which vector will be fetched when the channel is next serviced?
 - a) A Match Service Request vector because the Match occurred first.
 - b) A Match Service Request vector only if the MRL was set.
 - c) The Host Service Request vector because it has priority.
 - d) A vector determined by the combination of the Match and the Host Service Request.
 - e) None of the above
3. A new function is assembled with TPU2ASM and an S19 file has been created. To download the function to the 332 board, what order should these steps be taken?
 - a) Map and turn on the on-chip RAM, download the S19 file, set the EMU bit.
 - b) Download the S19 file to the on-chip RAM, set the EMU bit, map and turn on the on-chip RAM.
 - c) Set the EMU bit, Map and turn on the on-chip RAM, download the S19 file.
 - d) Map and turn on the on-chip RAM, set the EMU bit, download the S19 file.
 - e) None of the above
4. Consider the following TPU microcode sequence:

```
au    nil := a - diob, ccl;  
chan  set flag0.  
  
chan  clear flag1;  
if z = 0 then goto jump_ahead.  
  
chan  clear flag0.  
  
chan  set flag1.  
  
jump_ahead:
```

If A and DIOB were not equal when the sequence started, which of the following statements is TRUE about the channel flags at the label JUMP_AHEAD?

- a) Flag0 is set and Flag1 is cleared.
- b) Flag0 is cleared and Flag1 is set.
- c) Flag0 is set and Flag1 is set.
- d) Flag0 is cleared and Flag1 is cleared.
- e) None of the above

5. Consider the following code sequence:

```

au    p_low := p_low + 1.
ram   p <- @COUNTER;

au    p_high := 0.
ram   p -> @COUNTER;

```

What can be said of the contents of the parameter “COUNTER”?

- a) The high byte is cleared and the low byte is incremented.
- b) The high byte is cleared and the low byte is unchanged.
- c) The high byte is unchanged and the low byte is incremented.
- d) The parameter is unchanged.
- e) None of the above

6. Consider the following entry directive:

```

%ENTRY START_ADDR *; NAME = EXAM_COUNTER_ONE; DISABLE_MATCH;
COND HSR1 = 0, HSR0 = 0, LSR = 1, M/TSR = 1, PIN = X, FLAG0 = 1;
RAM p <- @COUNTER.

```

ASSUME this information to be true: the function is entered by a Match (MRL is set), a Transition (TDL is set), and a Link from another channel (LSL is set), all occurring before the channel is scheduled. Which of the following statements concerning the service latches is FALSE?

- a) The Match occurred before the Pin Transition.
- b) The Link must have been called by the channel that was serviced during the time slot immediately before this time slot.
- c) If the Match is serviced by microcode and only the MRL is negated, the function will re-enter the same state the next time the channel is serviced, assuming no host request is issued before the next service.
- d) If the Link is serviced by microcode and only the LSL is negated, the function will enter a new state the next time the channel is serviced, assuming no other link request is generated for this channel and no host request is issued before the next service.
- e) None of the above

7. Which of the following statements is FALSE concerning the time slot transition?

A time slot transition:

- a) Always takes 10 CPU clocks (not counting Ram Collision time or 4-clock NOPs).
- b) Happens after the service of a state on Channel X, before the service of a state on Channel Y.
- c) Happens after the service of a state on Channel X, before the service of another state on Channel X.
- d) The capture register of the channel about to be serviced is loaded into the ERT (event register temporary) during every time slot transition.
- e) None of the above

8. A system uses three channels: one High priority, one Middle priority, and one Low priority. Which of these statements is FALSE, assuming all three channels are constantly asking for service?

- a) The High priority channel will be serviced 4 times as much as the Low priority channel.
- b) A 4 CPU clock cycle NOP will be executed after every channel is serviced.
- c) A 4 CPU clock cycle NOP will be executed after every 7 time slots and not before.
- d) To find the Worst Case Latency for a channel, each running channel's longest state (not counting INIT states) must be included in the calculation.
- e) None of the above

9. Which of the following microinstructions is legal? Hint: look at your Quick Reference Guide.

- a) If Z = 0 then Goto Branch_Target, flush;
end.
- b) au ert := tcr1 + tcr2;
ram p -> @MY_PARAMETER.
- c) If true then Goto End_Of_Phase, flush;
chan enable_mtsr,
set FLAG0,
cir.
- d) au p := p - 1;
chan set FLAG0.
- e) None of the above

10. Consider the following microcode state:

```
%ENTRY START_ADDR *; NAME = WHAT_TO_MEASURE; DISABLE_MATCH;  
COND HSR1 = 0, HSR0 = 0, LSR = 0, M/TSR = 1, PIN = 1, FLAG0 = X;  
RAM DIOB <-@REFERENCE_TIME.  
  
au diob := ert - diob;  
ram diob -> @TIME_LGTH_MEASURED.  
au p := ert;  
ram p -> @REFERENCE_TIME.  
chan neg_tdl;  
end.
```


Besides an INIT state, this is the only state in the function.

What can be said about the application of this function?

- a) The function probably measures period.
- b) The function probably measures low time of a period.
- c) The function probably measures high time of a period (pulse width).
- d) The function probably will not work.
- e) None of the above

TPU FUNCTION DESCRIPTIONS



Available TPU Functions

Brief Description “contents.txt” (1 of 7)

CURRENT CONTENTS

DCPM Degree Clock with Period Measurement Size:79

Doc Source:BulletinB

This input function reads a signal from a toothed wheel, and inserts a selected number of counts at even intervals between the input teeth. The resulting count, driven on TCR2, continues to a fixed maximum and resets, providing an approximation of the angle of the wheel shaft. The period of the last two teeth is maintained in a 23 bit parameter. Stall indication is provided via a host interrupt.

DIO Discrete Input / Output Size:21

Doc Source:TPUrefMan

Allows a TPU channel to be used as a digital I/O pin. Using a transition detect, the function can indicate to the CPU when the pin has changed state when configured as an input. Additionally the host can setup a sample rate at which the pin will be tested and its state reported back to the CPU. The pinstate parameter contains the last 16 samples allowing debounce or noise rejection to be accomplished.

ITC Input Transition Counter Size:36

Doc Source:TPUrefMan

This function can perform the basic input capture operation where a timer value is captured on a pin transition and reported to the CPU. In addition, the user can specify a number of transitions to be counted. On each transition, the captured time is returned to the CPU and when the required number of transitions has been counted, the last transition time is stored in a different location and the CPU is interrupted. At this point the function can also automatically link to a number of other TPU channels. The sequence of counting transitions can be programmed to occur once only or continuously.

Available TPU Functions

Brief Description “contents.txt” (2 of 7)

OC Output Compare
 Doc Source:TPUrefMan

Size:39

This output function will generate a rising or falling edge or a toggle on the channel pin in one of two modes:- i) a single edge occurs a programmable delay from a user defined time - a start condition can also be specified allowing pulse generation. ii) on receiving a link from another channel, OC will generate a continuous 50% duty cycle waveform with a period that is calculated without CPU intervention by multiplying a period obtained on another TPU channel with a user specified ratio parameter. The OC function will also provide the CPU with the current TCR values.

PMA_PMM Period measurement missing/additional transition Size:88
 Doc Source:TPUrefMan

This input function, primarily designed for engine control, measures the period (in TCR1 clocks) between normal regularly occurring input transitions and makes this period available for use by other functions or the CPU. PMA mode will detect 'additional' transitions defined as being when the period between transitions is less than a user specifiable fraction of the last 'normal' period. PMM mode will detect 'missing' transitions defined as being when the period between transitions is greater than a user specifiable fractional multiple of the last 'normal' period. After a programmable number of 'additional/missing transitions' have been detected, the function will reset the TCR2 timer counter and interrupt the CPU. The PMA_PMM process is continuous requiring no CPU intervention once configured.

PPWA Period / pulse width accumulator
 Doc Source:TPUrefMan

Size:63

This input function accumulates a 24 bit sum of either the period or the high time pulse width of an input signal over a programmable number of periods/pulses (1 to 255). An interrupt to the CPU is generated when the specified number of periods/pulses has elapsed. The period/pulse width is sampled at a rate defined by the user and the incomplete sum is available to the user. By linking to an OC channel, an output signal can be generated that is proportional to that applied to the PPWA channel.



Available TPU Functions

Brief Description “contents.txt” (3 of 7)

PSP Position synchronised pulse Size:96

Doc Source:TPUrefMan

This output function generates pulses of variable length at specified 'angles', where an angle clock has been fed into the TCR2 input pin and its period measured in TCR1 clocks using the PMM or PMA function on another channel. The function uses a fractional multiply on the PMM/PMA period to accurately position pulses to a higher angular resolution than that provided by the TCR2 angle clock. The rising edge of the pulse is always defined as an angle and the falling edge can be specified as a second angle or a time after the rising edge. Once configured, the PSP function will continuously generate pulses based on the latest period information obtained from a PMM/PMA channel without CPU intervention.

PWM Pulse Width Modulation Size:32

Doc Source:TPUrefMan

This functions generates a pulse width modulated waveform in which the period and/or high time can be changed at any time by the CPU. The function supports both 0 and 100% duty cycles. Very high resolution can be obtained depending on the frequency required.

QDEC Quadrature decode Size:29

Doc Source:BulletinB

This input function uses two channels to decode a pair of out of phase signals in order to present the CPU with directional information and a position value. It is particularly suitable for use with the slotted encoders often employed in motor control, displacing expensive external decode solutions. The function derives full 4x resolution from the encoder signals and provides a 16 bit free running position counter. Additionally QDEC provides a time stamp for each quadrature edge, this can be used for position interpolation at very low speeds or in systems using a low resolution encoder.

Available TPU Functions

Brief Description “contents.txt” (4 of 7)

- QOM** Queued Output Match Size:49
Doc Source:BulletinB
This output function lets the user set up multiple output events in a table which will then be processed by the TPU independently of the CPU. The pin response for each output event is independently programmable allowing multiple entries with the same pin response to be used to effectively increase the range of a single output event. The function includes loop modes which will circle the table continuously or a programmable number of times. A link can also be used to fire the function.
- SM** Stepper Motor Size:71
Doc Source:TPUrefMan
This function allows a sequential group of up to 8 TPU channels to be used to drive a stepper motor. Once configured, the CPU specifies the required position and the function automatically accelerates and decelerates the motor to the desired position. The algorithm provides for linear acceleration and deceleration control of a motor with up to 14 programmable step rates.
- SPWM** Synchronised Pulse width modulation Size:54
Doc Source:TPUrefMan
This output function generates a PWM waveform in which the period and/or the high time can be changed at any time by the CPU. It is distinguished from the normal PWM function in that the rising edges have a programmable time relationship to transitions on another TPU channel which can be another SPWM channel. Three different modes allow complex timing relationships to be maintained between channels without CPU intervention. 0 and 100% duty cycles are not supported.
- SQW** Square wave generator Size:11
Doc Source:BulletinB
This output function generates a programmable frequency square wave. TCR1 is used as the timebase. The user specifies the half period of the output. Generation starts with a rising edge.

Available TPU Functions

Brief Description “contents.txt” (5 of 7)

RWTPIN Read/write timers and Pin Size:18
Doc Source:BulletinB
This function allows a TPU channel to be used as a digital I/O pin. It also allows the CPU to read both TCRs and to selectively write the TCRs. The function will receive links and on receipt of the link will read the timers, update the channel pin and interrupt the CPU. FC Jeff Wright
Update to include IC function - could then act as direction channel for UDC. The ability to write the TPU tcrs may not be very useful in practice, but it gives certain users a 'warm' feeling to have that degree of control.

PTA Programmable Time Accumulator Size:67
Doc Source:BulletinB
This input function accumulates a 32 bit sum of either the period, the high time or the low time of an input signal over a programmable number of periods/pulses (1 to 255). A CPU interrupt is generated (& optionally links) when the specified number of periods/pulses has elapsed. This function is an extension of the PPWA function, but without the programmable update rate.

RECTW Rectangular Wave Generator Size:14
Doc Source:BulletinB
This output function produces a continuous rectangular output waveform. The user specifies Period and High Time. The function can be thought of as a simple PWM output without 0 and 100% duty cycle support. The function uses TCR1 as a timebase and starts with a rising edge

RECTW2 Rectangular Wave Generator 2 Size:18
Doc Source:BulletinB
This output function produces either a continuous rectangular output waveform, or a single shot pulse. The user specifies Period and High Time. In continuous mode, the function can be thought of as a simple PWM output without 0 and 100% duty cycle support. The function uses TCR1 as a timebase. The user specifies the start pin condition.

Available TPU Functions

Brief Description “contents.txt” (6 of 7)

MHT Measure High Time Size:14

Doc Source:BulletinB

This input function measures the high time of and input signal and presents the CPU with the result in the form of a 16 bit TCR1 count. The functions operates continuously and issues an interrupt request to the CPU after every complete H_T measurement.

MHT2 Measure High Time 2 Size:15

Doc Source:BulletinB

This input function measures the high time of and input signal and presents the CPU with the result in the form of a 16 bit TCR1 count. The functions operates continuously and issues an interrupt request to the CPU after every complete H_T measurement. After the first complete high time measurement after initialisation, the function also issues a link request to a user specified channel.

SIOP Serial Input/Output Port Size:39

Doc Source:BulletinB

This function uses 2 or 3 TPU channels to implement a uni or bi-directional synchronous serial port that can be used to communicate with a wide variety of devices. Features such as baud rate and transfer size are programmable. The function can also produce a clock only, when it uses just one channel.

UART Universal Asynchronous Receiver/Transmitter Size:66

Doc Source:BulletinB

This function uses 1 or 2 TPU channels to implement a uni or bi-directional asynchronous serial port. The function supports a variable word length up to 14 bits and even/odd/no parity generation and checking. The baud rate is freely programmable and can be as high as 200Kbaud depending on other TPU activity. The function is very similar in use to a standard Motorola SCI port.

Available TPU Functions

Brief Description "contents.txt" (7 of 7)

"FORTHCOMING ATTRACTIONS" - Availability of these new routines is POSSIBLE, NOT Guaranteed.

PAPW - Pulse Accumulate in Programmable Window

This function will count the number pulses in a user defined window on a single shot or continuous basis. Makes for easy frequency measurement.

PTAL - Programmable Time Accumulator with Links

Identical to PTA except that links are generated to a user defined group of channels at the end of the accumulation.

MCPWM - Multi Channel PWM

This function will use multiple TPU channels to produce high speed centre aligned PWMs suitable for a variety of motor control. The function will include programmable dead time for H-bridge driving.

COMM - Commutation Function

This function will use multiple TPU channels to produce the commutation drive signals for a variety of brushless motors. It will use the position count generated by QDEC as the basis for commutation generation. Motors supported will include 3 and 4 phase brushless d.c. and 3 phase switched reluctance.

TSM - Table Stepper Motor

This function will be an improved version of the current SM function with a user defined acceleration profile in a table and a improved positioning strategy.

ICTC - Input Capture Transition Counter

This is a development of the original ITC function and is very similar in operation. A new feature allows any parameter RAM location to be "captured" instead of TCR1 or 2.

UDC - Up / Down Counter

This function can be used as an alternative to QDEC. It uses a clock and a direction signal. Since the clock is fed into the TCR2 input pin, a very fast count rate is possible.

