Chapter 10
Interrupts

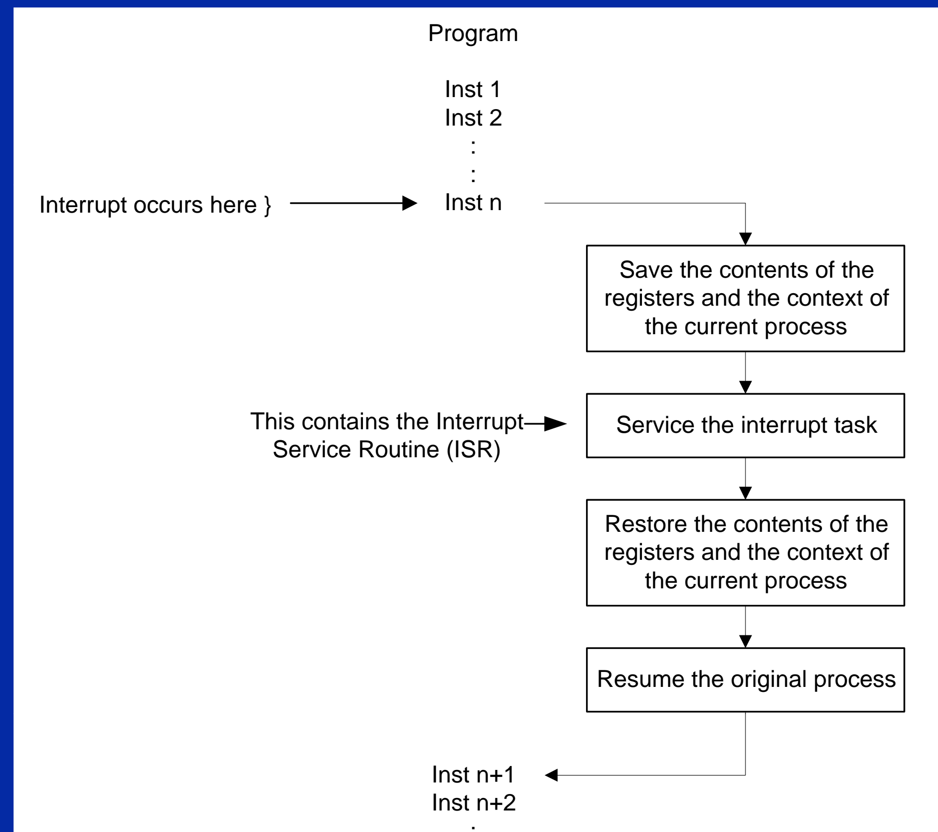
Learning Objectives

- Introduction to interrupts.
- Types of interrupts and sources.
- Interrupt timeline.
- Handling and processing interrupts using C and assembly code.

 Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Introduction

- Interrupts are used to interrupt normal program flow so that the CPU can respond to events.
- The events can occur at anytime.



Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

CPU Interrupts and Sources

Sources	Description
(HPI) DSPINT	HPI Interrupt
TINT0	Timer 0
TINT1	Timer 1
SD_INT	SDRAM Refresh
EXT_INT4	External Interrupt 4
EXT_INT5	External Interrupt 5
EXT_INT6	External Interrupt 6
EXT_INT7	External Interrupt 7
DMA_INT0	DMA Channel 0
DMA_INT1	DMA Channel 1
DMA_INT2	DMA Channel 2
DMA_INT3	DMA Channel 3
XINT0	McBSP Channel 0 TX
RINT0	McBSP Channel 0 RX
XINT1	McBSP Channel 1 TX
RINT1	McBSP Channel 1 RX

High

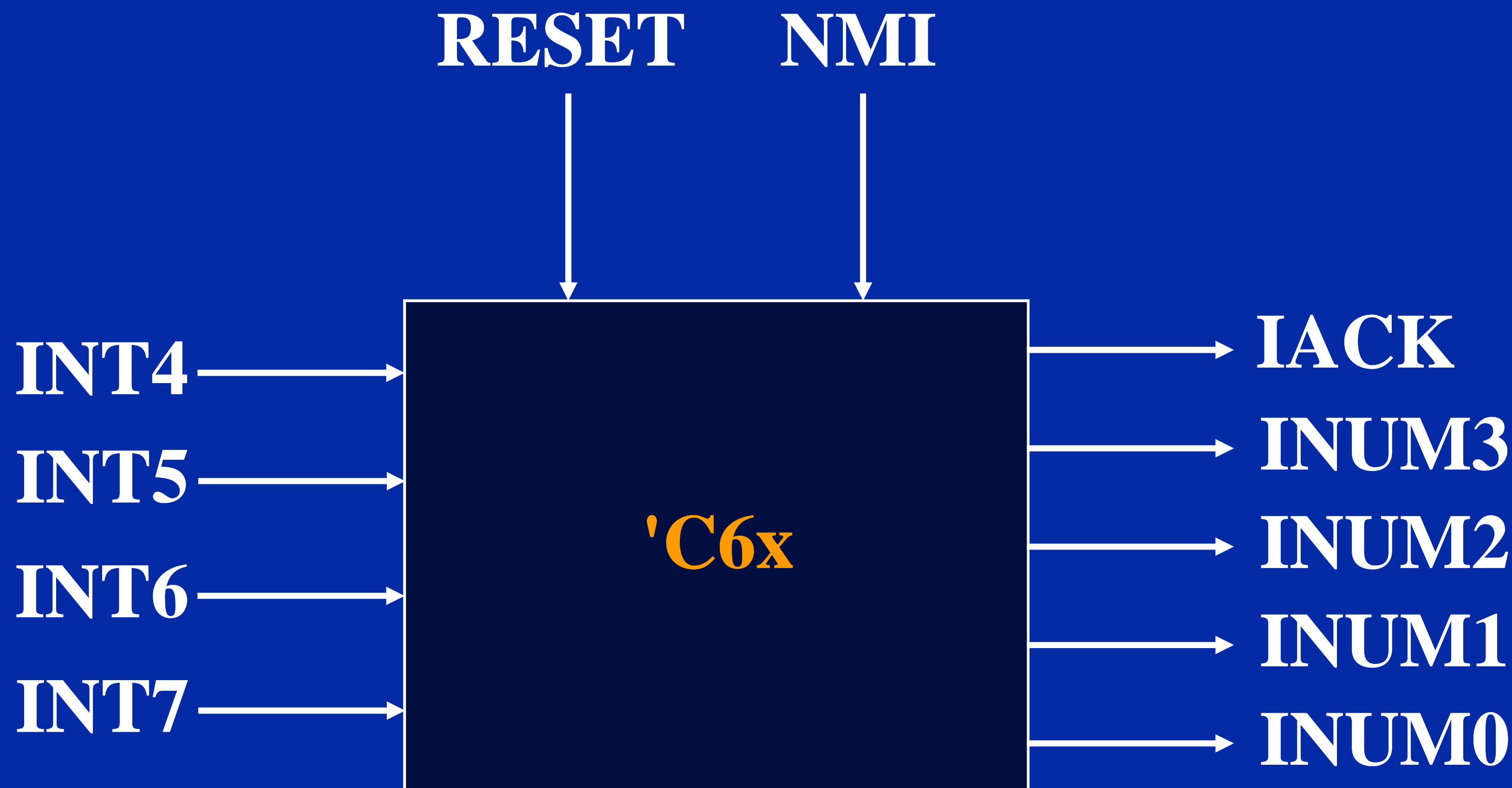
Priority

Low

Note that there are more sources of interrupt than the CPU can handle.

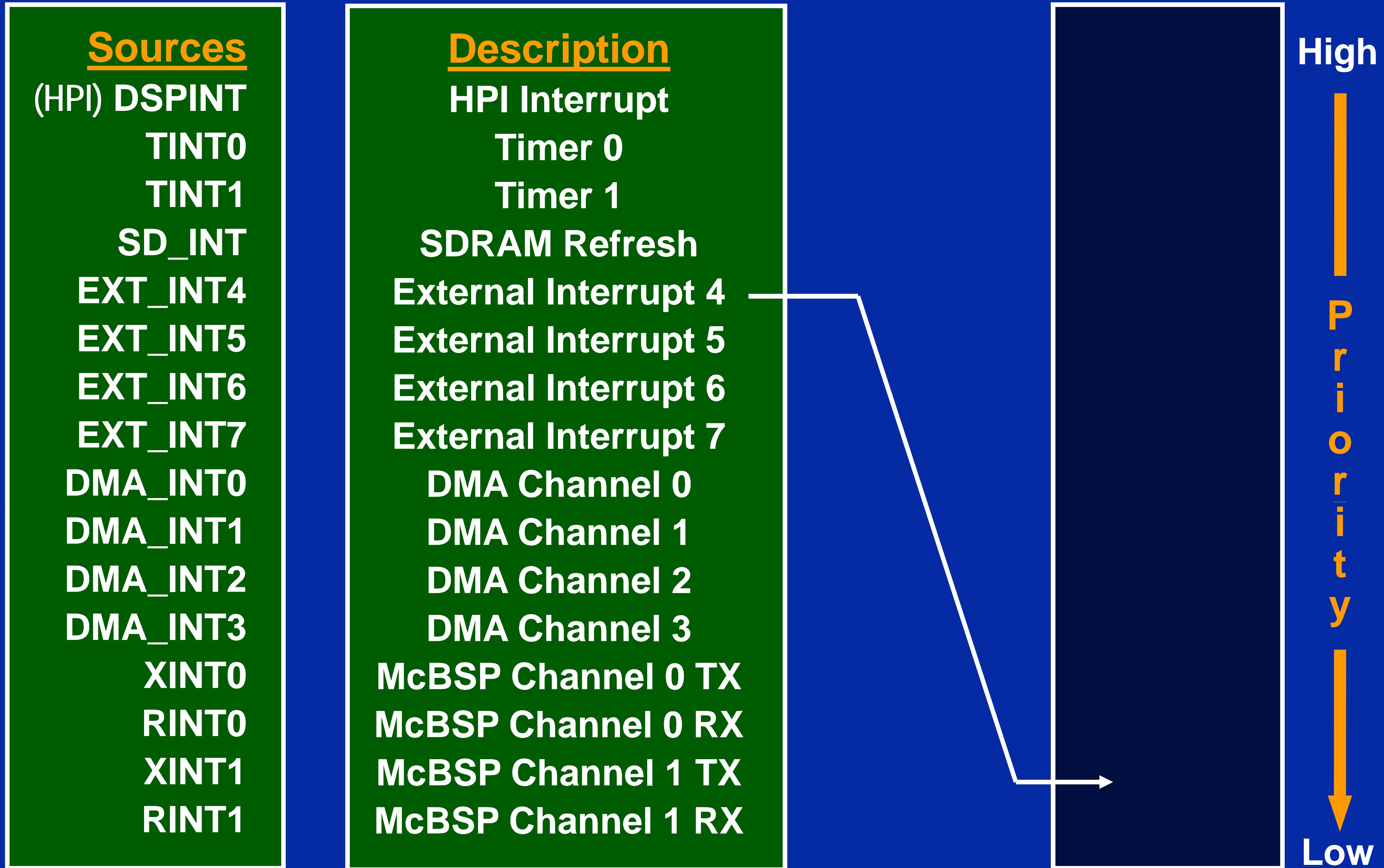
Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

CPU Interrupts and Sources



- The IACK and INUM pins do not exist on the C621x, C671x and C64x devices.

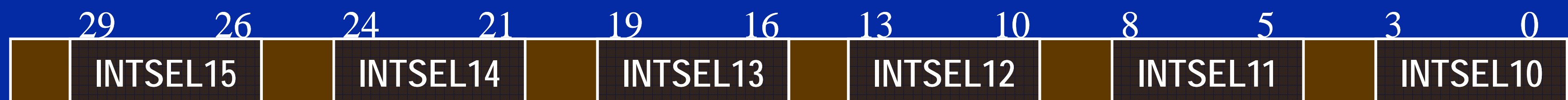
Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002


Interrupt Selection
Sources
(HPI) DSPINT
TINT0
TINT1
SD_INT
EXT_INT4
EXT_INT5
EXT_INT6
EXT_INT7
DMA_INT0
DMA_INT1
DMA_INT2
DMA_INT3
XINT0
RINT0
XINT1
RINT1
Description
HPI Interrupt
Timer 0
Timer 1
SDRAM Refresh
External Interrupt 4
External Interrupt 5
External Interrupt 6
External Interrupt 7
DMA Channel 0
DMA Channel 1
DMA Channel 2
DMA Channel 3
McBSP Channel 0 TX
McBSP Channel 0 RX
McBSP Channel 1 TX
McBSP Channel 1 RX
High
Priority
Low
Each source interrupt can be made to trigger a specific CPU interrupt.

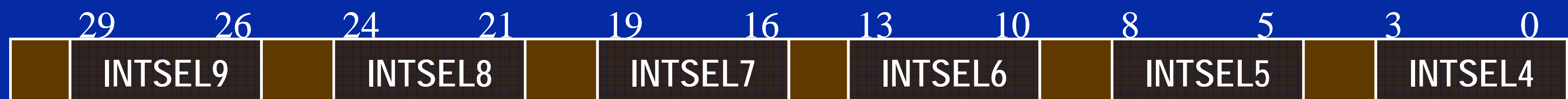
Interrupt Selection

- The interrupt source mapping can be achieved by initialising the appropriate bit-fields of the interrupt multiplexer.
- Each CPU interrupt has a selection number "INTSEL#" that specifies the source.

Interrupt Multiplexer High(INT10 - INT15) (address 0x19c0000)



Interrupt Multiplexer Low(INT4 - INT9) (address 0x19c0004)

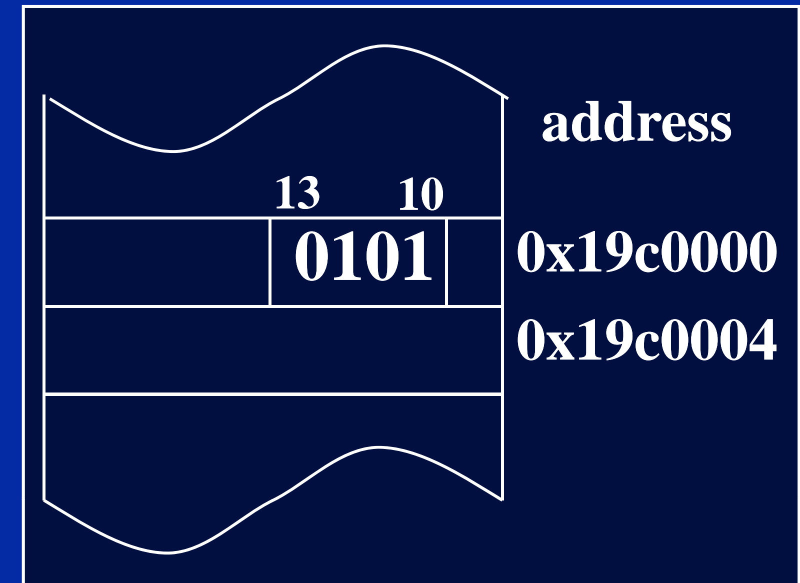


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

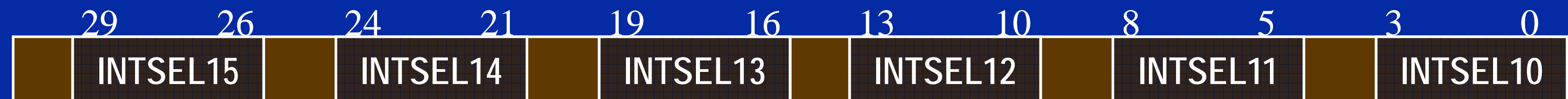
Interrupt Selection

- **Example: Mapping EXT_INT5 to CPU INT12.**

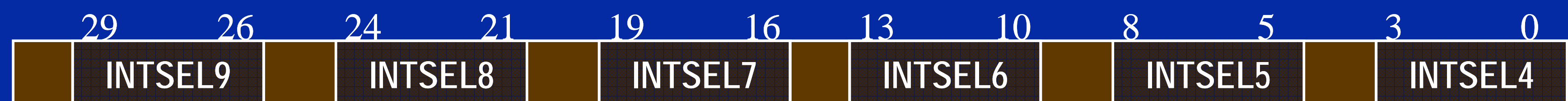
Write 5_{dec} = 0101b to INTSEL12



Interrupt Multiplexer High (INT10 - INT15) (address 0x19c0000)



Interrupt Multiplexer Low (INT4 - INT9) (address 0x19c0004)



Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Interrupt Selection

- **Writing code to initialise INT12 with 0101b, there are 3 methods:**

  (1) Using assembly:

```
MVKL    0x19c0000, A1
MVKH    0x19c0000, A1
LDW     *A1, A0
CLR     A0, 10, 13, A0
SET     A0, 10, 10, A0
SET     A0, 12, 12, A0
STW     A0, *A1
```

  (2) Using the Chip Support Library:

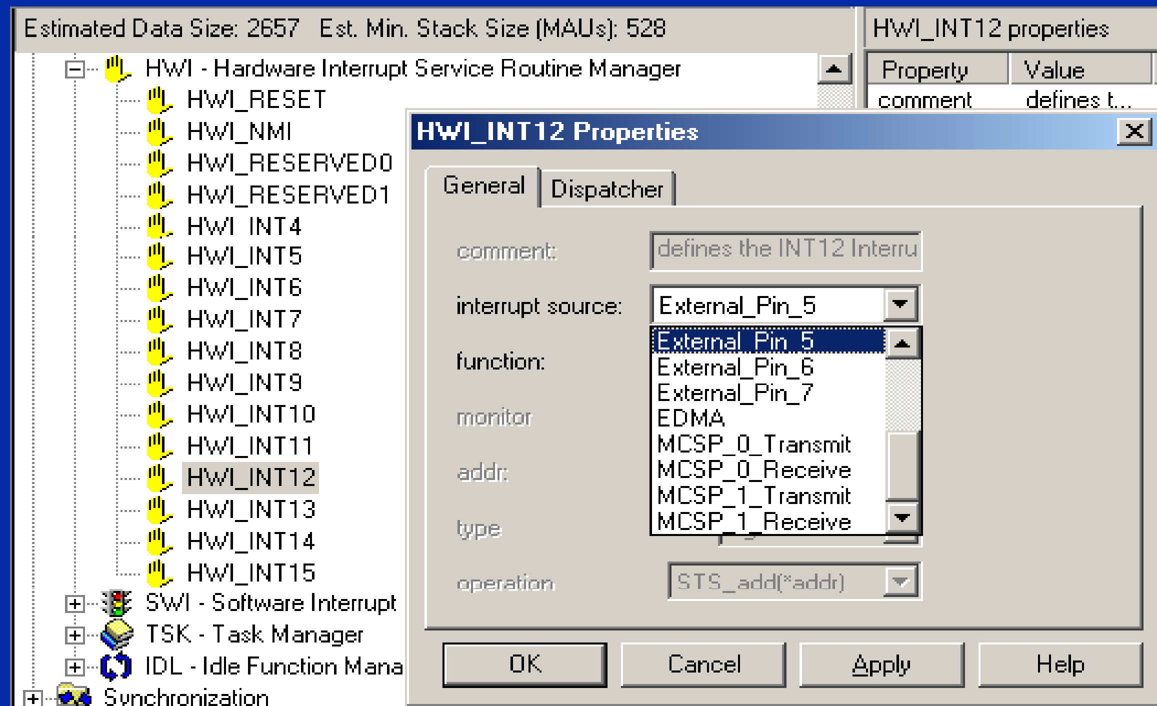
```
#include <intr.h>
#include <regs.h>


IRQ_map (IRQ_EVT_EXTINT5, 12);
```


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Interrupt Selection

(3) Using the GUI interface:

- Open the CDB file.
- Select Hardware Interrupt Service Routine Manager.
- Select the CPU interrupt 12 (HWI_INT12) and right click and select properties.
- Select External_Pin_5 as the interrupt source.



Interrupt Timeline

User Responsibility (Initialisation)

Configure

1. Select interrupt sources and map them.
2. Create interrupt vector table.

Enable

3. Enable individual interrupts.
4. Enable global interrupt.

Performed by the CPU

5. Check for valid signal.
6. Once a valid signal is detected set flag bit.
7. Check if interrupt is enabled, if yes branch to ISR.

User Responsibility (Algorithm)

8. Write context store routine.
9. Write the ISR.
10. Write context restore routine.
11. Return to main program.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(2) Creating an Interrupt Vector

- When an interrupt occurs the CPU automatically recognises the source of the interrupt and jumps to the interrupt vector location.
- In this location a program is found which instructs the processor on the action(s) to be taken.
- Each vector location can accommodate eight instructions which correspond to a fetch packet.
- Such a location is know as the Interrupt Service Fetch Packet (ISFP) address.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(2) Creating an Interrupt Vector

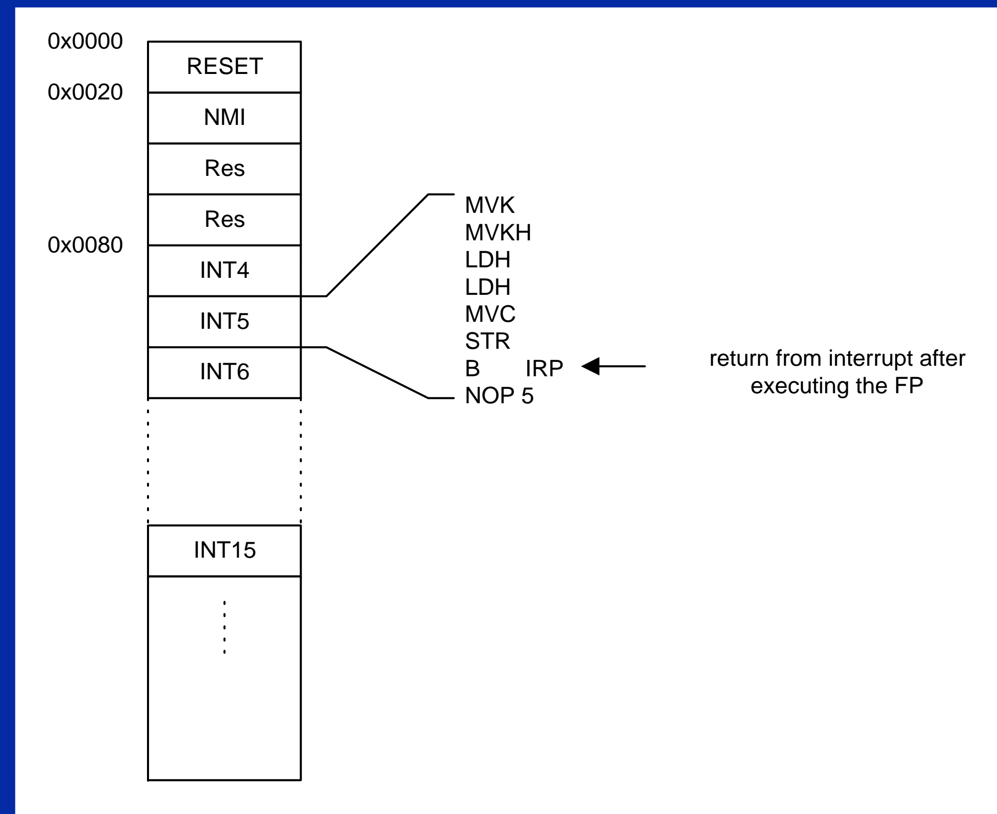
- The following table shows the interrupt sources and associated ISFP addresses:

Interrupt service table (IST)	
Interrupt Sources	**ISFP Addresses**
Reset	0x0000
NMI	0x0020
Reserved	0X0040
Reserved	0X0060
INT4	0X0080
INT5	0X00A0
INT6	0X00C0
INT7	0X00E0
INT8	0X0100
INT9	0X0120
INT10	0X0140
INT11	0X0160
INT12	0X0180
INT13	0X01A0
INT14	0X01C0
INT15	0X01E0

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

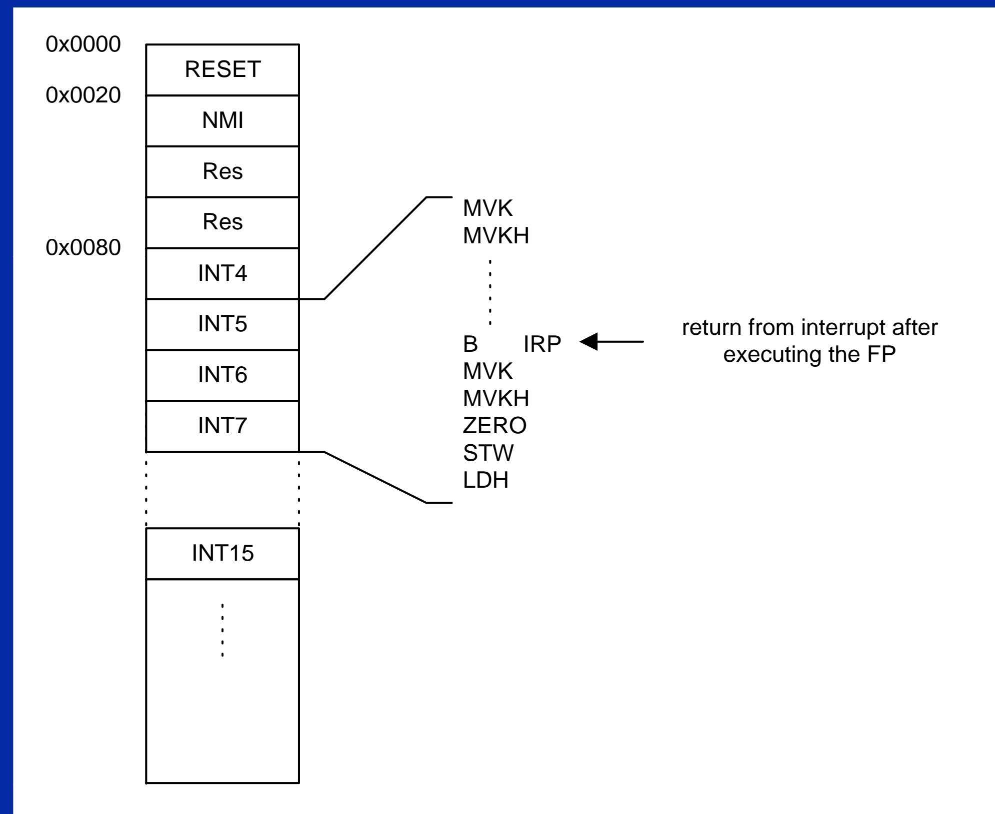
(2) Creating an Interrupt Vector

- **Example 1: ISR fits into a single Fetch Packet (FP).**



(2) Creating an Interrupt Vector

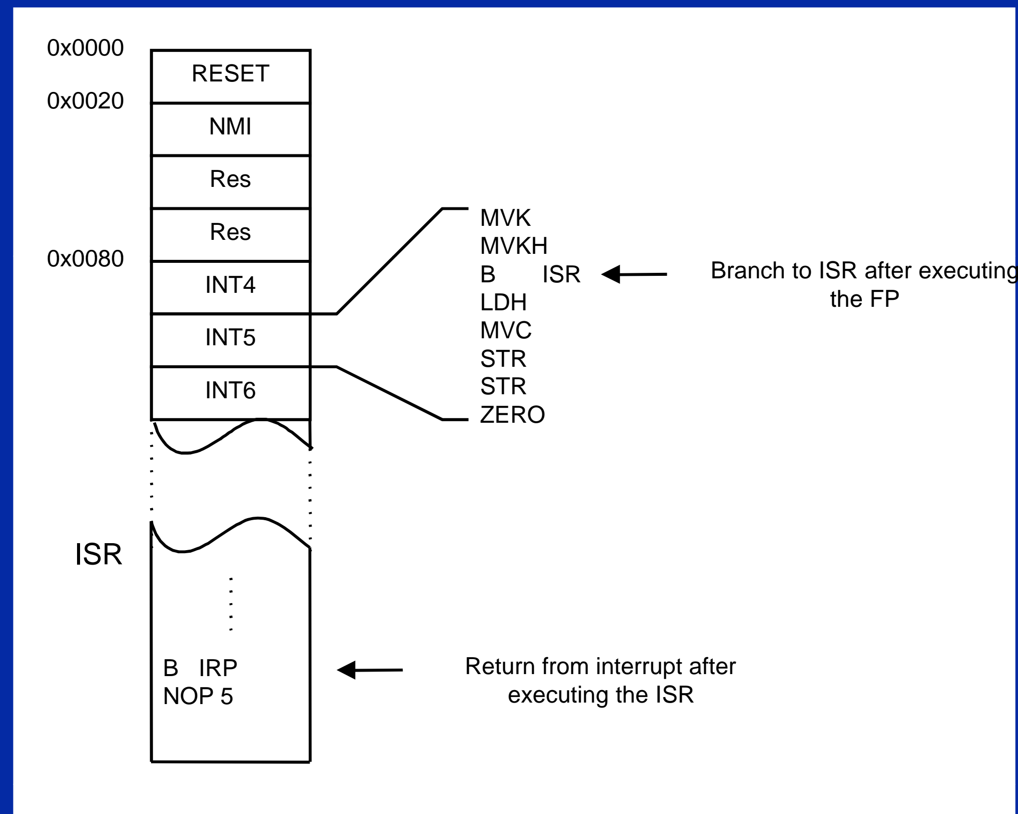
- **Example 2: ISR fits into multiple successive FP's (assuming the next interrupts are not used).**



Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(2) Creating an Interrupt Vector

- **Example 3: ISR is situated outside the interrupt vector table.**



Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(2) Relocating the Vector Table

- In general the vector table or the section "vectors" is linked to address zero.
- However in many applications there is a need to change the location of the vector table.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(2) Relocating the Vector Table

- This is due to many factors such as:
  - Moving interrupt vectors to fast memory.
  - Having multiple vector tables for use by different tasks.
  - Boot ROM already contained in memory starting at address 0x0000.
  - Memory starting at location zero is external and hence there will be a need to move the vector table to internal memory to avoid bus conflict in shared memory system.
- In order to relocate the vector table, the Interrupt Service Table Pointer (ISTP) register should be set up.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(2) Relocating the Vector Table

Interrupt service table (IST)	
Interrupt Sources	**ISFP Addresses**
Reset	ISTB + 0x0000
NMI	ISTB + 0x0020
Reserved	ISTB + 0X0040
Reserved	ISTB + 0X0060
INT4	ISTB + 0X0080
INT5	ISTB + 0X00A0
INT6	ISTB + 0X00C0
INT7	ISTB + 0X00E0
INT8	ISTB + 0X0100
INT9	ISTB + 0X0120
INT10	ISTB + 0X0140
INT11	ISTB + 0X0160
INT12	ISTB + 0X0180
INT13	ISTB + 0X01A0
INT14	ISTB + 0X01C0
INT15	ISTB + 0X01E0

(2) Relocating the Vector Table

- **Example: Relocate the ISTP to address 0x800.**

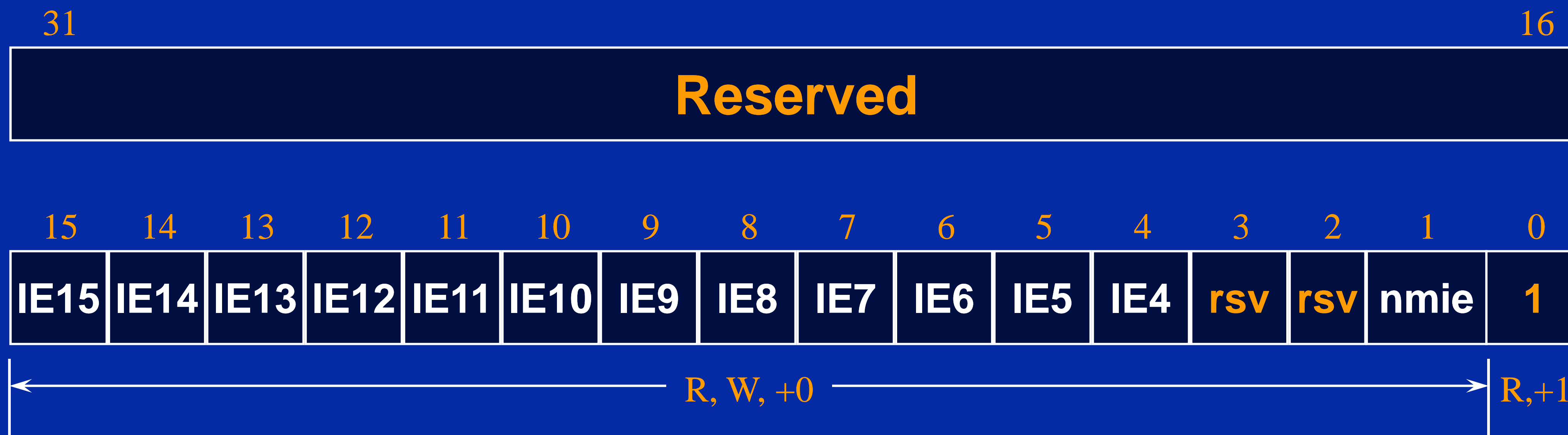
(1) The interrupt vector code located between 0x000 and 0x200 must be copied to the location 0x800 and 0x800 + 0x200 (0xA00).

(2) Initialise the ISTP.

```
MVKL    0x800, A0
MVKH    0x800, A0
MVC     A0, ISTP
```


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(3) Enable Individual Interrupts



- To enable each int, write "1" to IE bit
- IER bits are NOT affected by the value in global interrupt enable (GIE)

- **Example: Write some code to enable INT7.**

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(3) Enable Individual Interrupts

- **Method 1: Using "C" code.**

```
#include <c6x.h>
void enable_INT7 (void)
{
        IER = IER | 0x040;
}
```

- **Method 2: Using assembly.**

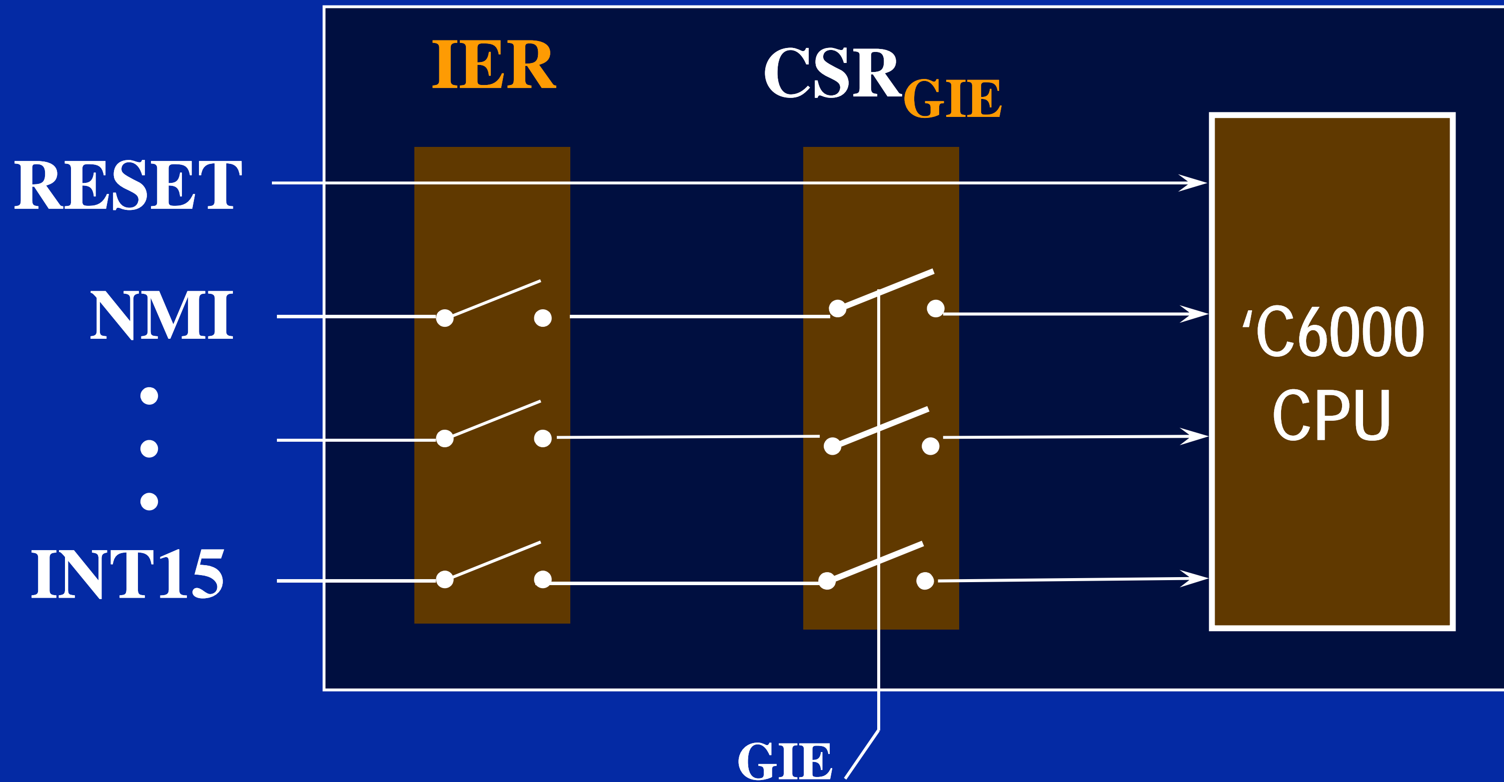
```
_asm_set_INT7
        MVC .S2 IER, B0
        SET .L2 B0,7, 7, B0
        MVC .S2 B0, IER
```

- **Method 3: Using the CSL.**

```
#include <csl.h>
#include <csl_irq.h>

IRQ_enable (IRQ_EVT_EXTINT7)
```


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(4) Enable Global Interrupt



- IER allows the enabling or disabling of interrupts individually.
- GIE bit allows the enabling or disabling of interrupts globally (all at once).

Note: By disabling the GIE interrupts can be prevented from occurring during initialisation.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(4) Enable Global Interrupt

- Method 1: Using "C" code.

```
#include <c6x.h>
void enable_GIE (void)
{
        CSR = CSRIER | 0x1;
}
```

- Method 2: Using assembly.

```
_asm_set_GIE
        MVC .S2 CSR, B0
        SET .L2 B0,0, 0, B0
        MVC .S2 B0, CSR
```

- Method 3: Using the CSL.

```
#include <csl.h>
#include <csl_irq.h>

IRQ_globalEnable ()
```


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(4) Enable Global Interrupt RESET & NMI

Reset: is not maskable. IER[0] = 1

NMI: once the NMI is enabled it will be non maskable.

- NMIE bit enables the NMI.

So why have a non maskable interrupt that can be masked?

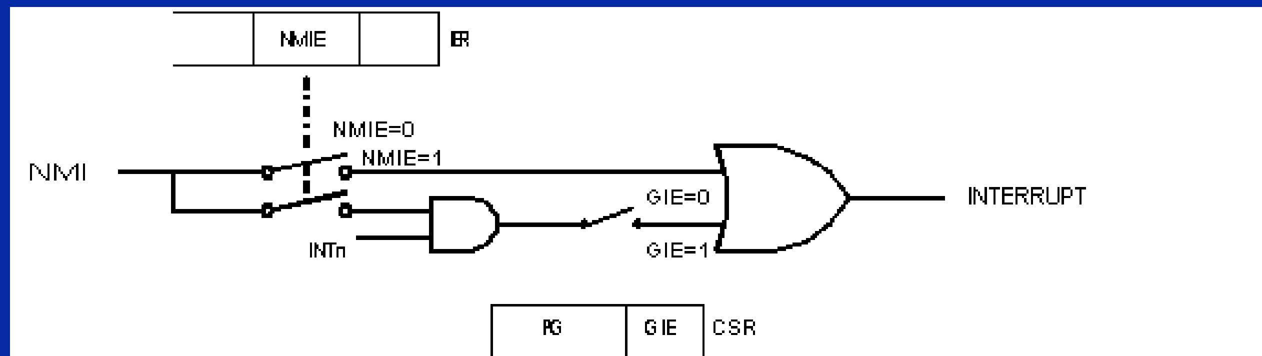
- Avoids unwanted NMI interrupts occurring between the time of a reset and the end of initialisation.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(4) Enable Global Interrupt RESET & NMI

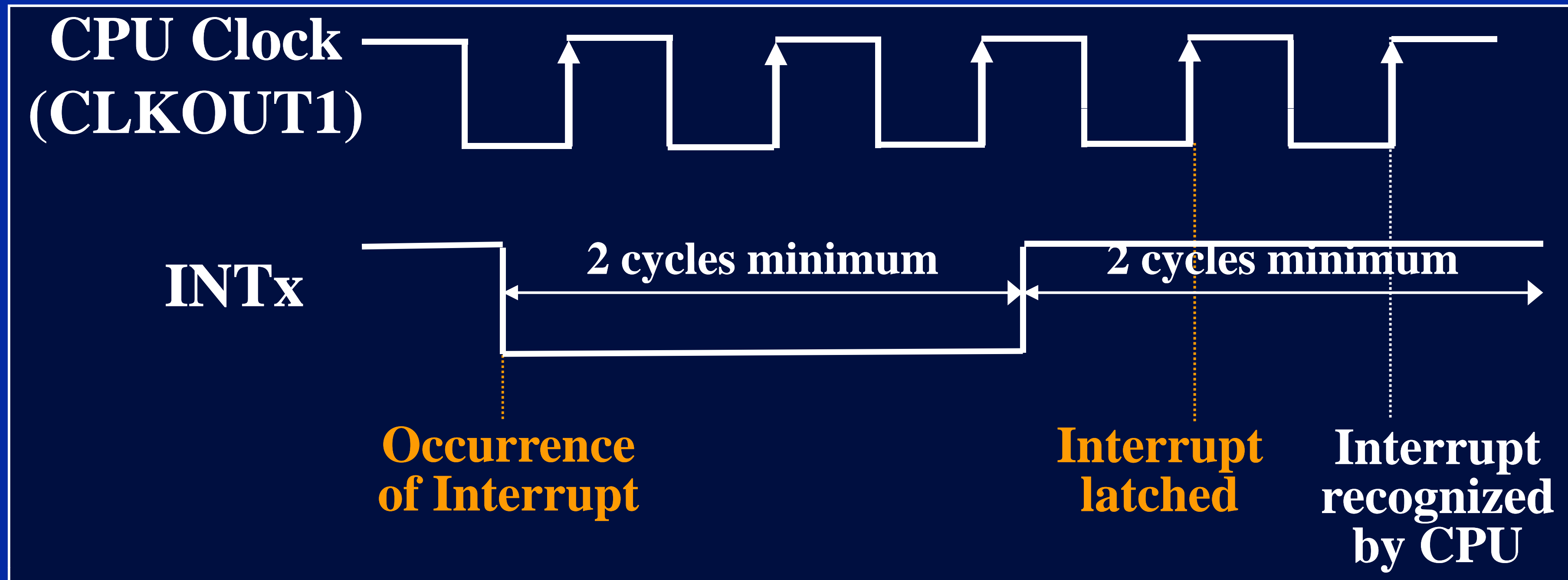
So how is the non maskable interrupt made maskable?

- **Once the NMI is enabled it cannot be disabled.**
- **Also the NMI must be enabled before any other interrupt (except reset) can be activated.**



Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(5) Check for a Valid Signal



Conditions for recognition of an external interrupt:

- **The interrupt must be held low for at least 2 cycles then high for at least two cycles.**

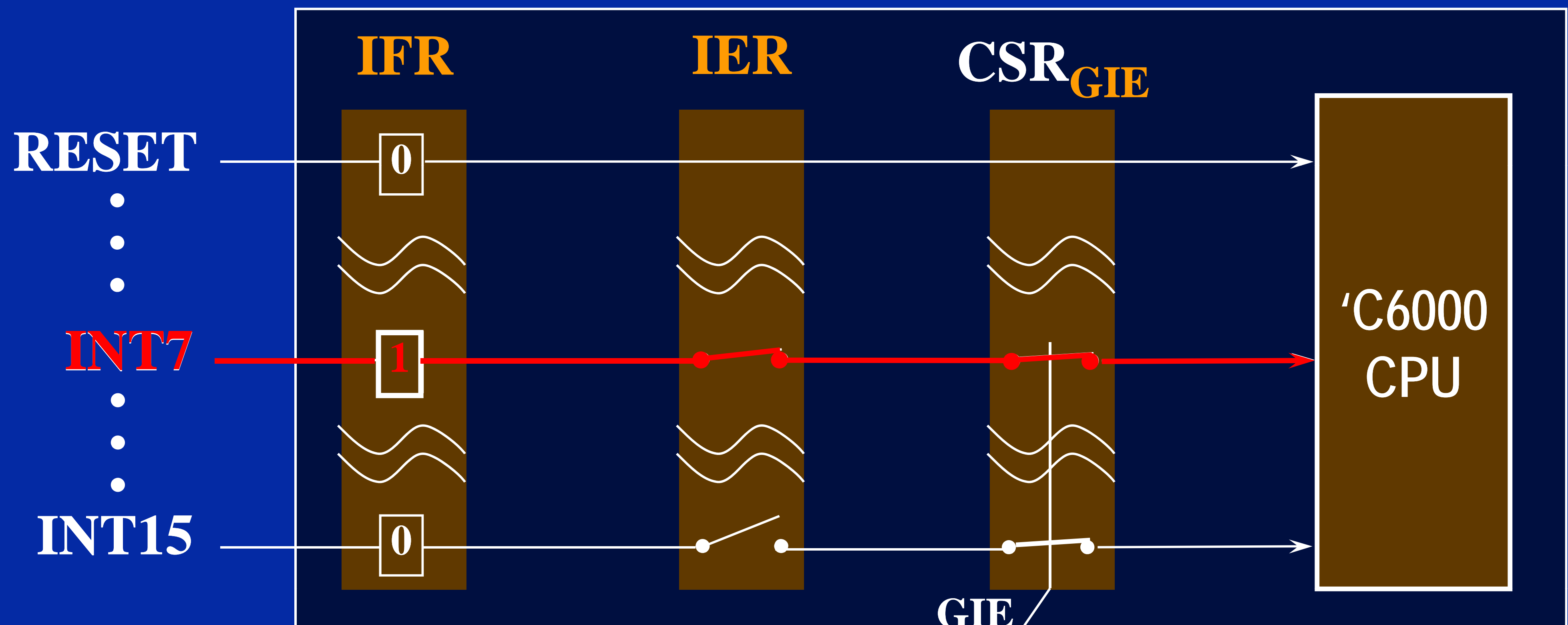
Recognition of the interrupt:

- **The interrupts are latched on the rising edge of CLKOUT1.**
- **The interrupt is finally recognised by the CPU one cycle after being latched.**

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(6) Interrupt Flag Register (IFR)

- **When an interrupt is recognised the corresponding bit in the IFR is set:**
  - **e.g. if INT7 is recognised then IFR[7] bit is set.**



Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(7) CPU Interrupt Hardware Sequence

What does the CPU do when an interrupt is recognised?

CPU Action	Description
0 → IFR (bit)	Clears corresponding interrupt flag bit
GIE → PGIE	Save previous value of GIE
0 → GIE	Disables global interrupts
Save next EP address	Save return address in IRP/NRP
Vector (ISTP) → PC	Loads PC with interrupt vector addr
1 → IACK pin	IACK is asserted
INUM(0-3)	INUM pins display corresponding int

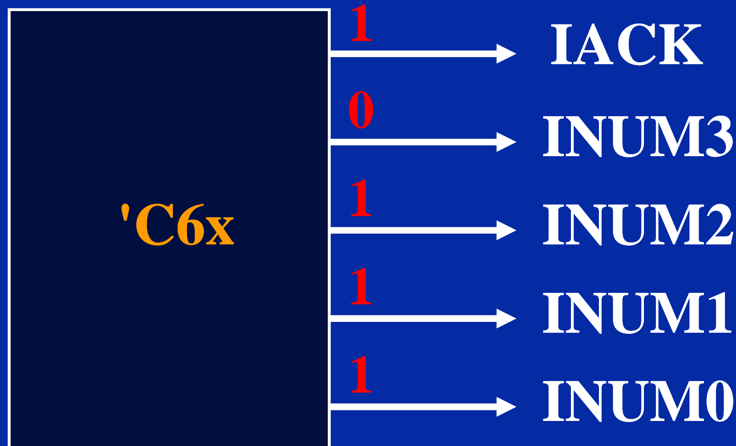
IACK and INUM pins are only available on the C620x and C670x.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(7) CPU Interrupt Hardware Sequence

e.g. if INT7 is recognised and IER[7] is set:

```
INT7 ->   0x0800000    ADD A0, A1, A2
          0x0800004    MPY A2, A6, A7
```



IACK and INUM pins are only available on the C620x and C670x.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(8) Writing the ISR in C

There are two methods of declaring an ISR:

(1) Traditional method:

```
interrupt void ISR_name (void);

main (void)
{
...
}
interrupt void ISR_name (void)
{
...
}
```

Notes:

- You need to use the interrupt keyword in order to inform the compiler that it is an ISR and therefore to handle the necessary register preservation and use the IRP for returning from the interrupt.
- No arguments can be passed to the ISR.
- No argument can be returned.

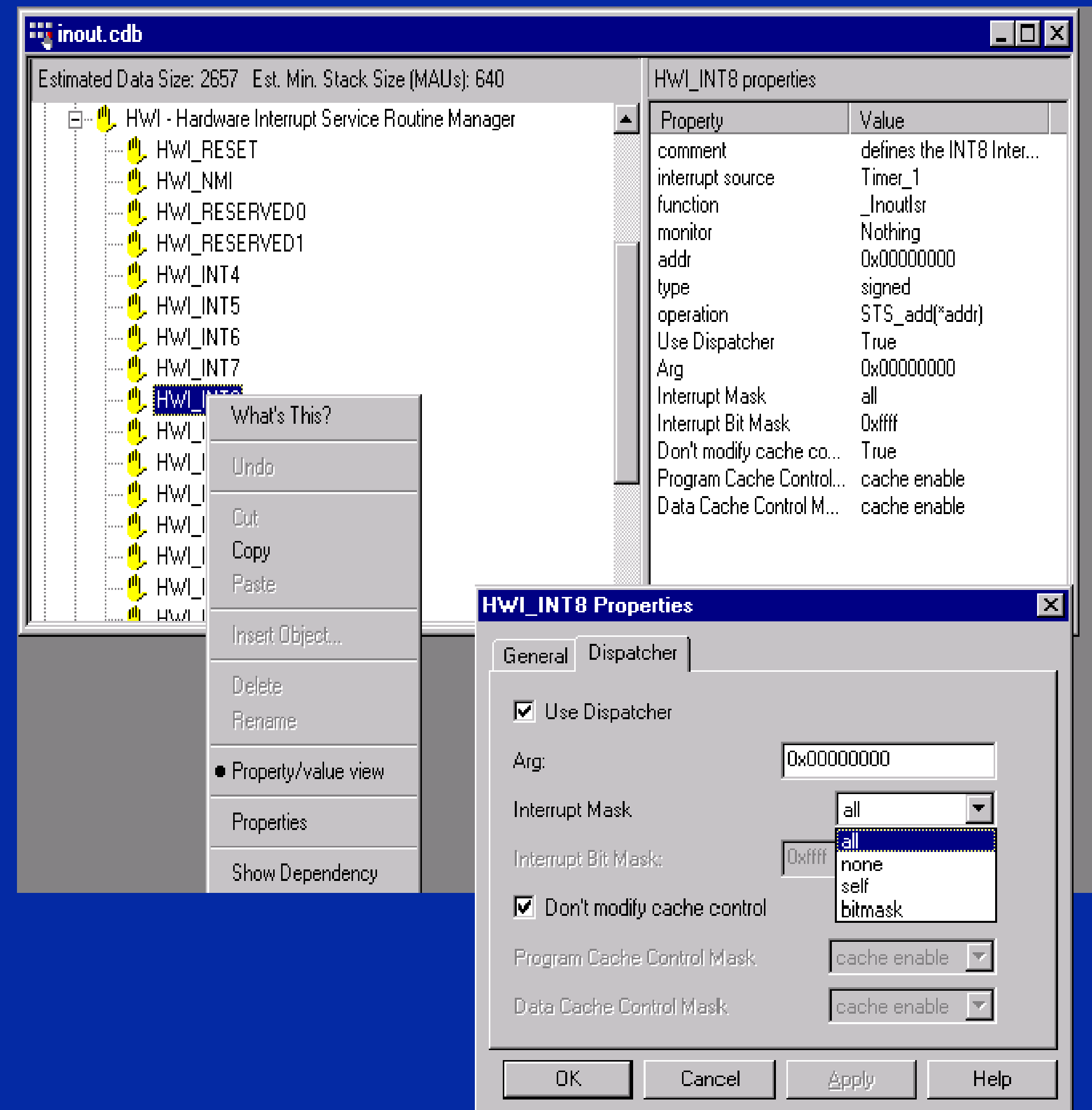
Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(8) Writing the ISR in C

(2) Using the dispatcher in DSP/BIOS:

Notes:

- You do not need to use the interrupt keyword.
- Interrupt nesting of interrupt functions written in “C” is allowed.
- You can pass one argument.
- You can specify which interrupts can be nested by specifying a mask.



 Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(8) Writing the ISR in Assembly

- **For maskable interrupts the return from interrupt address is always stored in the Interrupt Return Pointer (IRP).**
- **For non-maskable interrupts the address is stored in the Non-maskable Return Pointer (NRP).**

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

(8) Writing the ISR in Assembly

```
_isr_function

        stw
        .           (1) Save any registers used by the ISR on the stack
        .           (2) Use the HWI macros, or
        .           (3) Use the hardware interrupt dispatcher

        ldh
        .
        .           Put you ISR code here
        .

        ldw
        .           (1) Restore the registers saved on the stack
        .           (2) Use the HWI macros, or
        .           (3) Use the HWI dispatcher

        b irp       Return
        nop 5       (You can also use HWI macros)
```

It is much simpler to use the HWI dispatcher rather than doing the save and restore of registers by yourself or the HWI enter and HWI_exit macros.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Other Related Topics

(1) Setting and clearing the Interrupt Flag Register (IFR):

- Why do you need to read or write to the IFR?
  - By writing you can simulate an interrupt.
  - By reading you can check (poll) if an interrupt has occurred.
- You cannot directly write to the IFR:
  - To set a bit you have to write to the Interrupt Set Register (ISR).
  - To clear a bit you have to write to Interrupt Clear Register (ICR).

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Other Related Topics

- Example: Set and clear bits 7 and 8.

```
MVKL    0x0180, A0
MVC     A0, ISR
```

A0

ISR

* One cycle later → IFR

```
MVKL    0x0180, A0
MVC     A0, ICR
```

A0

ICR

* One cycle later → IFR

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Other Related Topics

(2) Setting the polarity of the External Interrupts:

- The External Interrupt Polarity (XIP) register allows the polarity of the external interrupt pins to be changed.
- This prevents extra hardware (space and cost) being required if the source has a different polarity.

0 = low to high (default)

1 = high to low (inverted)

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Other Related Topics

There are 3 ways of programming the XIP:

- **Method 1: Using "C" code.**

```
* (unsigned volatile int*) _IRQ_EXTPOL_ADDR = 2;
```

- **Method 2: Using assembly.**

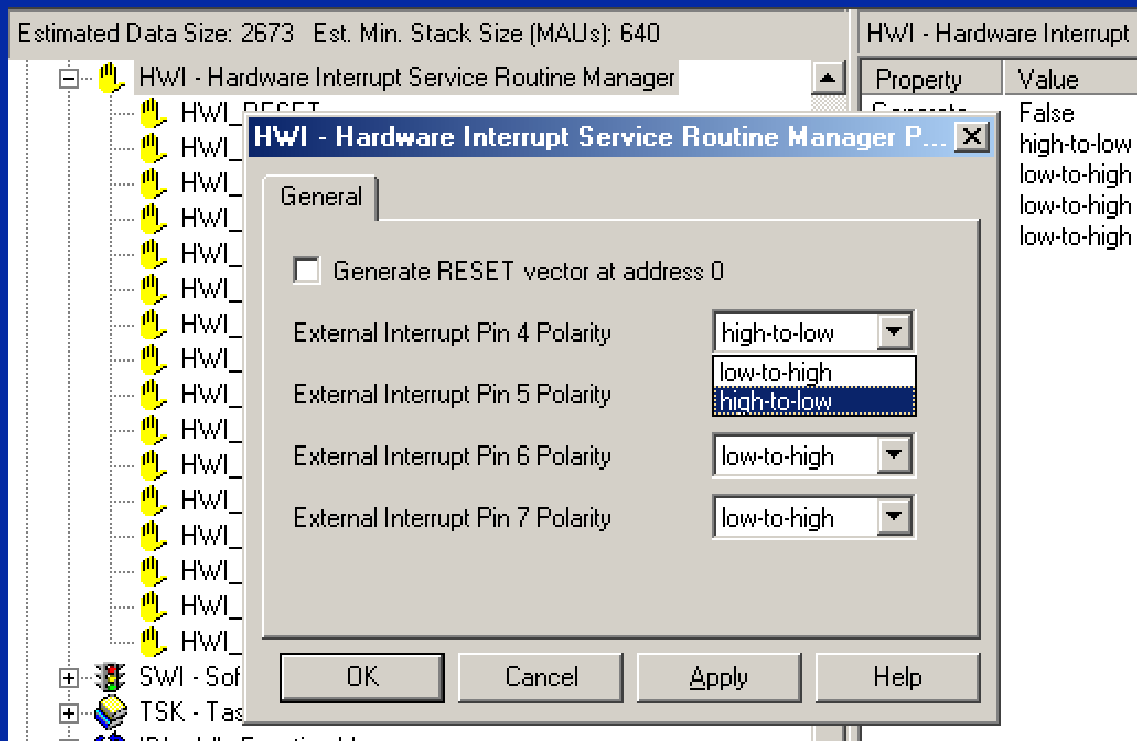
```
MVKL    2, A1
MVKL    0x19c0008, A0
MVKH    0x19c0008, A0
STW     A1, *A0
```


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Other Related Topics

There are 3 ways of programming the XIP:

- **Method 3: Using the GUI configuration tool.**



Note: The XIP only affects interrupts to the CPU and has no effect on the EDMA events.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Programming Interrupt Examples

- **Interrupts are programmed in the code in the following chapters:**
  - **\Code\Chapter 14 - Finite Impulse Response Filters**
  - **\Code\Chapter 15 - Infinite Impulse Response Filters**
  - **\Code\Chapter 16 - Adaptive Filters**
  - **\Code\Chapter 17 - Goertzel Algorithm**
  - **\Code\Chapter 18 - Discrete Cosine Transform**
  - **\Code\Chapter 19 - Fast Fourier Transform**

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Chapter 10
Interrupts
- End -

Single and Multiple Assignment

Single and Multiple Assignment

- **Single assignment** requires that no registers are read which have pending results.
- **Multiple assignment** allows multiple values to *re-use* the same register in different time slots (sort of register based"TDM") - thus reducing register pressure.

Single Assignment:

```
      MVKH  .S1  0x02,A1
SA:   B     .S1    SA
      LDW   .D1    *A0,A1
      NOP          4
      MPY   .M1    A1,A2,A3
      NOP
      SHR   .S1    A3,15,A3
      ADD   .L1    A3,A4,A4
```

Reads current value -- var(n)

Uses current value -- var(n)

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Single and Multiple Assignment

- **Single assignment** requires that no registers are read which have pending results.
- **Multiple assignment** allows multiple values to *re-use* the same register in different time slots (sort of register based"TDM") - thus reducing register pressure.

Multiple Assignment:

```
      MVKH .S1 0x02,A1
MA:   B        .S1   MA
      LDW      .D1   *A0,A1
      MPY      .M1   A1,A2,A3
      NOP
      SHR      .S1   A3,15,A3
      ADD      .L1   A3,A4,A4
```


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Single and Multiple Assignment

- **Single assignment** requires that no registers are read which have pending results.
- **Multiple assignment** allows multiple values to *re-use* the same register in different time slots (sort of register based"TDM") - thus reducing register pressure.

Single Assignment:

```
        MVKH   .S1  0x02,A1
SA:     B      .S1    SA
        LDW    .D1    *A0,A1
        NOP           4
        MPY    .M1    A1,A2,A3
        NOP
        SHR    .S1    A3,15,A3
        ADD    .L1    A3,A4,A4
```

Multiple Assignment:

```
        MVKH   .S1  0x02,A1
MA:     B      .S1    MA
        LDW    .D1    *A0,A1
        MPY    .M1    A1,A2,A3
        NOP
        SHR    .S1    A3,15,A3
        ADD    .L1    A3,A4,A4
```


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Problem with Multiple Assignment

Multiple Assignment:

```
        MVKH  .S1  0x02,A1
MA:     B          .S1     MA
        LDW        .D1     *A0,A1
        MPY        .M1     A1,A2,A3
        NOP
        SHR        .S1     A3,15,A3
        ADD        .L1     A3,A4,A4
```

Interrupt occurs here → (LDW)

- If an interrupt occurs at the LDW, this instruction will be completed before the MPY is executed.
- Therefore A1 will be loaded by the value pointed by A0 which will be used by the MPY instruction.
- The result is that A1 will NOT be equal to 0x02 as intended.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Chapter 10
Interrupts
- End -