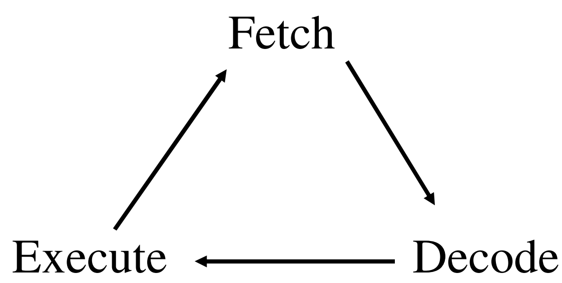
# ECE 344
# Microprocessors

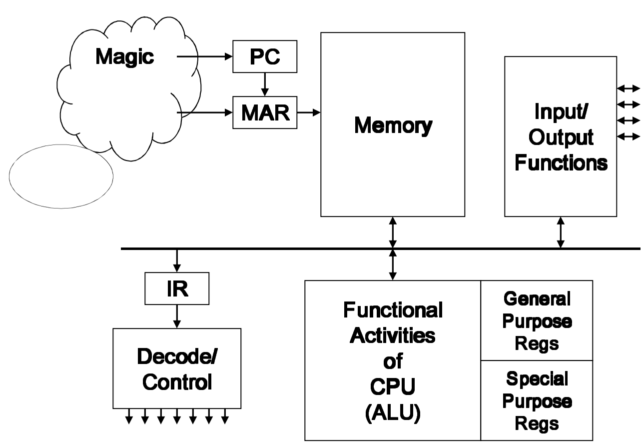
## How to Do Work in Processor

- Instruction set defines what's possible
- Work accomplished by repeating instructions on data until desired result has been obtained

# Basic Premise of All Computers




# Basic Computer Organization



## Computer Architecture

- Work is done in computers by moving information from one place to another, sometimes with a transformation
  - Register to register
  - Register to memory
  - Memory to register
  - Memory to memory

## Computer Architecture

- The instruction set identifies the work (movement, transformation) that can be done in a particular architecture
- Any work to be done in a machine must be accomplished by a series of individual instructions
- Complexity of instruction set determines how much work can be done by an instruction

## Complex Instruction Set Computer : CISC

- Instructions complex to allow for extended work
- Memory to memory activities allowed
- Instructions may take multiple words to store
- Complexity may limit pipelining
- Hooks to help with data structures

## Reduced Instruction Set Computer : RISC

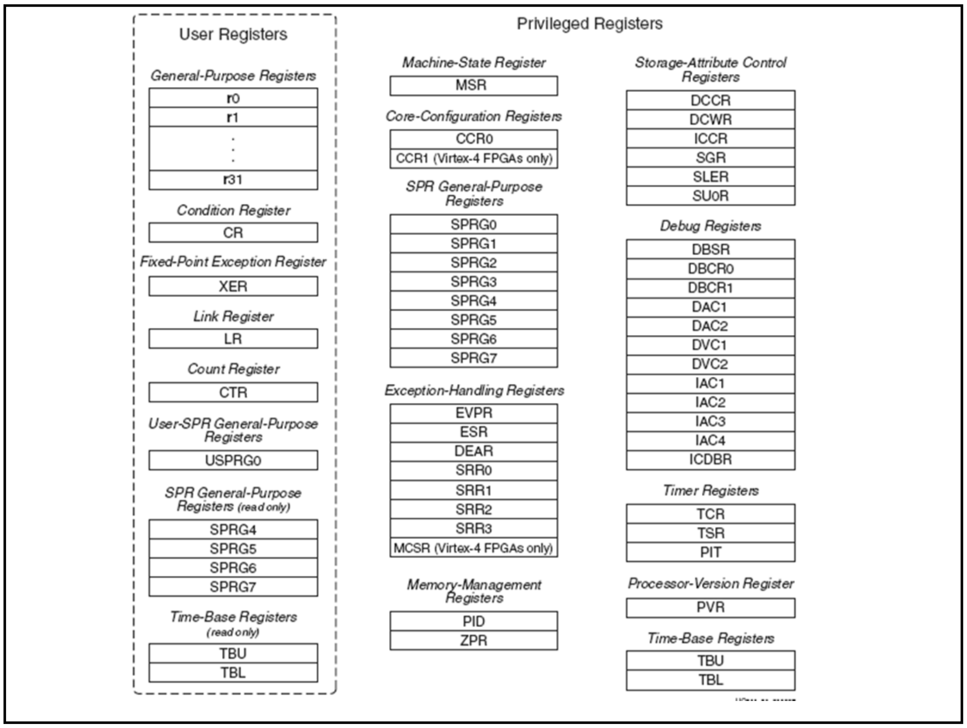
- Minimal number of instructions and addressing modes
- Fixed instruction format
- Hardwired instruction decoding
- Single cycle execution
- Load/store organization (work in registers)
- Use of register windows

## Common Registers in Processors

- Program counter (PC) – where is program executing
- Memory Address Register (MAR) – location of interest in memory
- Accumulator (ACC) – for single address machine
- General purpose registers (Rn) – for data and addresses

## Common Registers in Processors

- Address registers (An) – point to location in memory
- Instruction Register (IR) – holds instruction to be executed
- Status Register (SR) – holds information about status of system
- System Control Registers – hold information about overall system operation




# Processor Function in Register Form

Example: add r10, r11,r12

*fetch:* MAR ← PC

IR ← M [ MAR ]

PC ← PC + ??

*- decode -*

*execute:* r10 ← r11 + r12

# Instruction Types

- Work: add, subtract, mult, div, AND, OR, EXOR
- Movement: reg to reg, reg to mem, mem to reg
- Program control: jump, jump to subroutine, return from subroutine, conditional jump
- System control: set control bits, modify control registers, interrupt system

# System Memory

- Used to hold Instructions, Data, OS info
- Organized by byte (8 bits), halfword (16 bits), or word (32 bits)
- Instructions can move bytes, halfwords, or words to or from memory
- Registers identify location in memory (either PC or Rn)

# Memory Organization - Bytes

| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
|---|---|---|---|---|---|---|---|---|
| 0000 | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| 0008 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| 0010 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 0018 | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F |

# Memory Organization - Halfwords

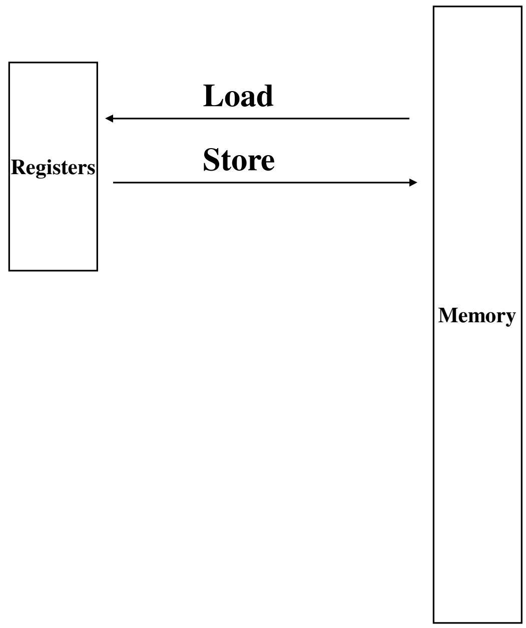
| | 00 | 02 | 04 | 06 |
|---|---|---|---|---|
| 0000 | 0001 | 0203 | 0405 | 0607 |
| 0008 | 0809 | 0A0B | 0C0D | 0E0F |
| 0010 | 1011 | 1213 | 1415 | 1617 |
| 0018 | 1819 | 1A1B | 1C1D | 1E1F |

# Memory Organization - Words

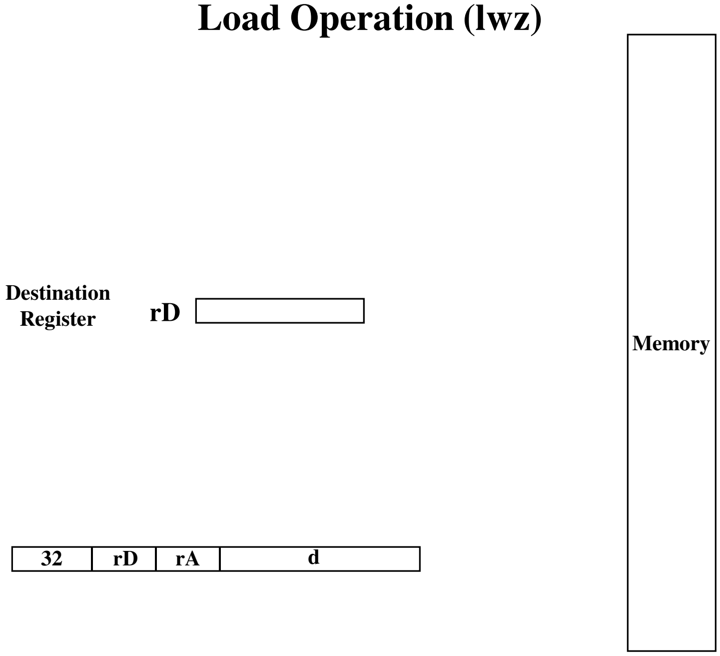
| | 00 | 04 |
|---|---|---|
| 0000 | 00010203 | 04050607 |
| 0008 | 08090A0B | 0C0D0E0F |
| 0010 | 10111213 | 14151617 |
| 0018 | 18191A1B | 1C1D1E1F |

# Load & Store: Use Register as Pointer

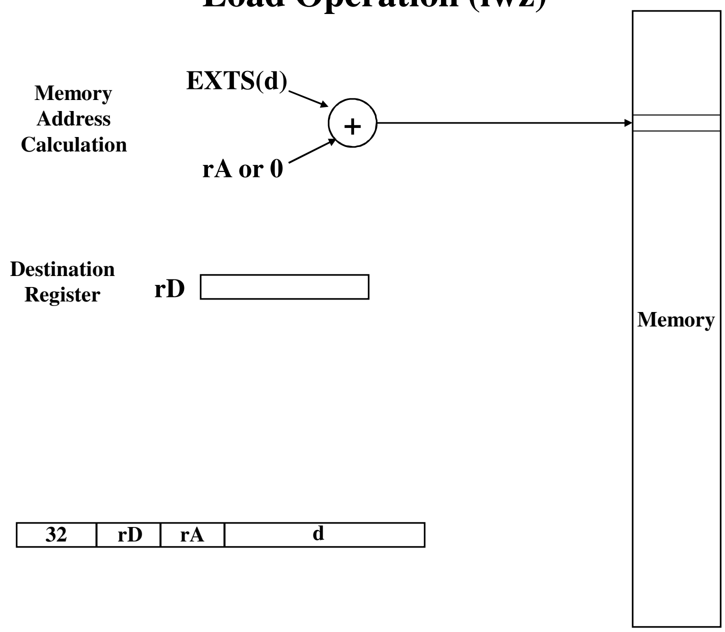
- Goal: move data to (load) or from (store) register file
- Specify one register as pointer to memory
- Specify another register as target
- Specify offset in terms of bytes



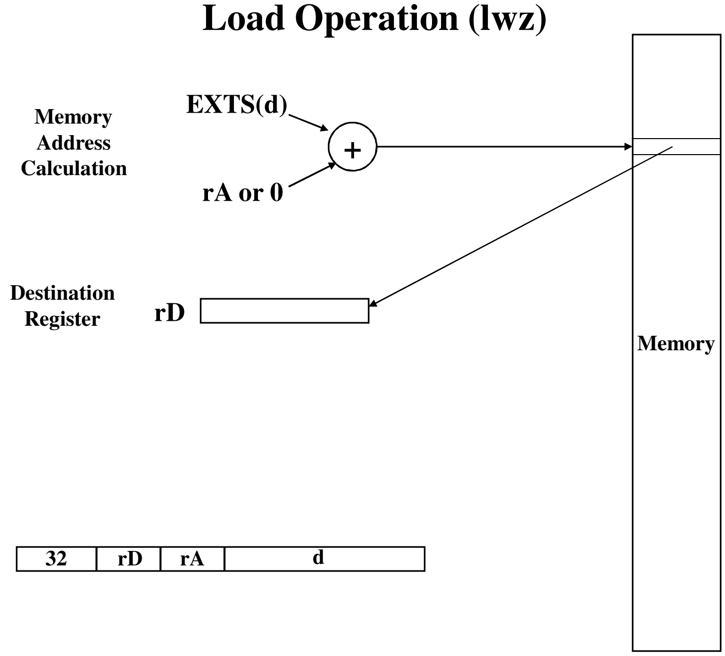

# Load Operation (lwz)



# Load Operation (lwz)




# Load Operation (lwz)



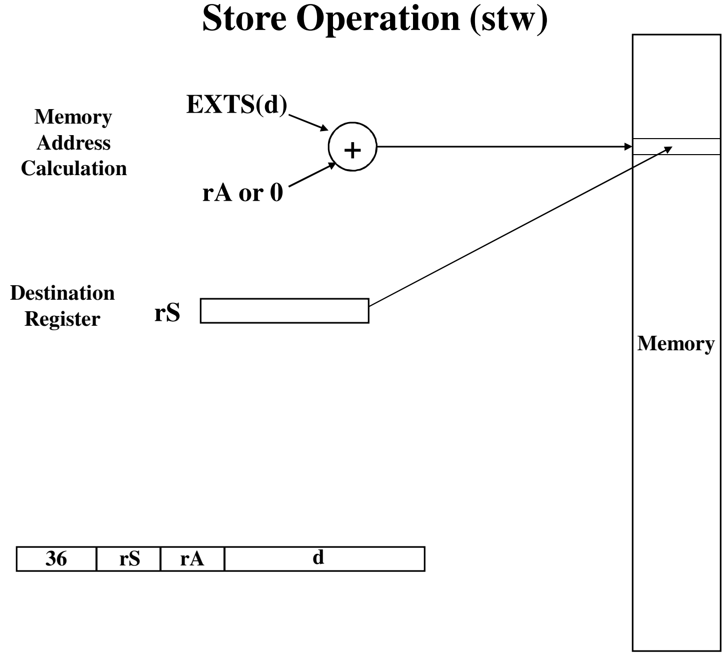
## Load Operation (lwz)

*Fetch:* MAR ← PC
IR ← M [ MAR ]
PC ← PC + 4

*Decode:*

*Execute:* EA ← (rA or zero) + EXTS(d)
rD ← M [ EA ]

## Store Operation (stw)



## Subroutine call: Program Control plus Data Movement

- Program Control: Determine target and transfer control to that location (BL, others)
- Subroutine linkage: return address → Link Register
- Parameter passing: agreement between calling and called routine
- Simple parameter passing: agree on register usage
- More complex parameter passing: stack

## Software Stuff

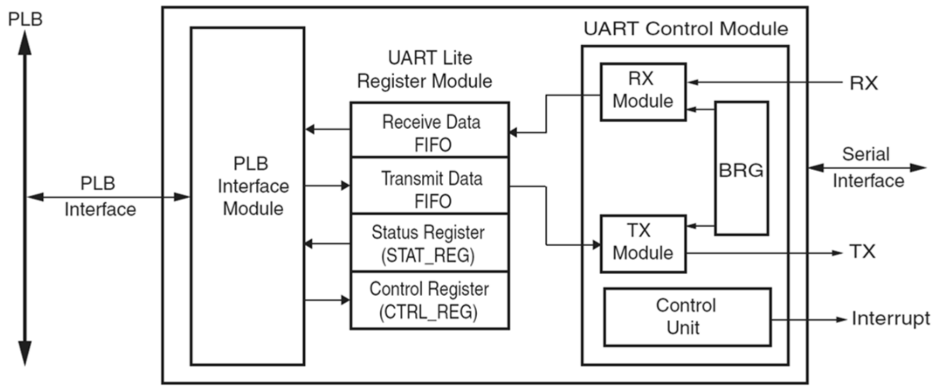
- Array implementation and utilization
  - One dimension – single index
  - Multiple dimension – calculate offset, then single index
- Other data structures
  - Stack
  - Queue
  - List
- Loops and conditional execution

## Memory Mapped I/O

- I/O elements assigned addresses in memory space of processor
- I/O activity carried out by writing/reading locations in memory address space
- Work can be accomplished by bits, bytes, halfwords, words
- Understanding of I/O function imperative

## UART: Serial Transmission

- Universal Serial Receiver/Transmitter (UART)
- Functionality of serial activity: specification and activity
- Interrupt activities for variety of events




## XPS UART Lite Register Descriptions

Table 4 shows all the XPS UART Lite registers and their addresses.

*Table 4:* **XPS UART Lite Registers**

| Base Address + Offset (hex) | Register Name | Access Type | Default Value (hex) | Description |
|---|---|---|---|---|
| C_BASEADDR + 0x0 | Rx FIFO | Read | 0x0 | Receive Data FIFO |
| C_BASEADDR + 0x4 | Tx FIFO | Write | 0x0 | Transmit Data FIFO |
| C_BASEADDR + 0x8 | STAT_REG | Read | 0x4 | UART Lite Status Register |
| C_BASEADDR + 0xC | CTRL_REG | Write | 0x0 | UART Lite Control Register |

# Timer Module

- Two different modes – capture and generate
- One or two timers per module
- Three registers per counter
  - Counter itself
  - Load register
  - Control/Status register
- Interrupt capability

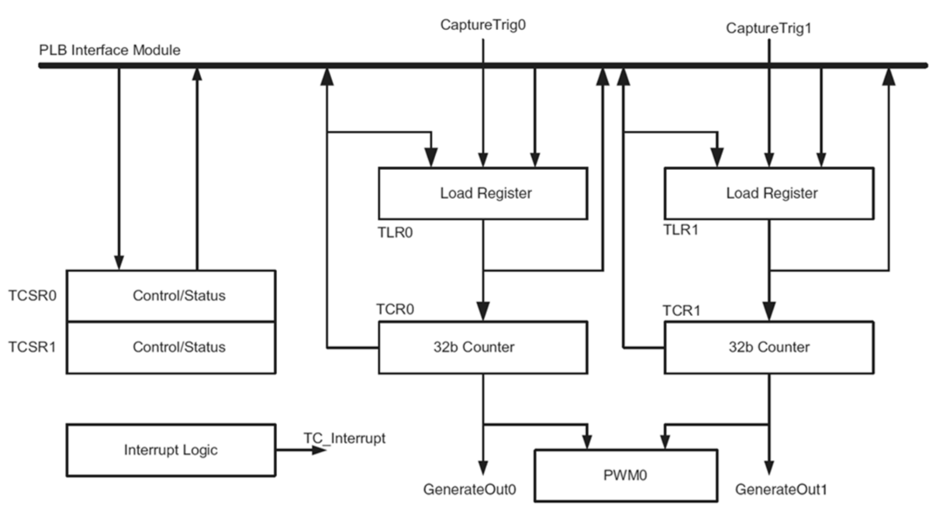


*Table 4:* **XPS Timer/Counter Register Address Map**

| Register | Address (Hex) | Size | Type | Description |
|---|---|---|---|---|
| TCSR0 | C_BASEADDR + 0x00 | Word | Read/Write | Control/Status Register 0 |
| TLR0 | C_BASEADDR + 0x04 | Word | Read/Write | Load Register 0 |
| TCR0 | C_BASEADDR + 0x08 | Word | Read | Timer/Counter Register 0 |
| TCSR1 | C_BASEADDR + 0x10 | Word | Read/Write | Control/Status Register 1 |
| TLR1 | C_BASEADDR + 0x14 | Word | Read/Write | Load Register 1 |
| TCR1 | C_BASEADDR + 0x18 | Word | Read | Timer/Counter Register 1 |

# Pulse Width Modulation (PWM)

- Variable amount of power delivered to pin based on register settings
- Period: how often pulse happens
- Width: how much of period signal asserted
- Control: what pins used for what, when

# GPIO Module

- General Purpose I/O
  - In, Out, or In & Out
- Can have one or two interfaces per module
- Can use with interrupt

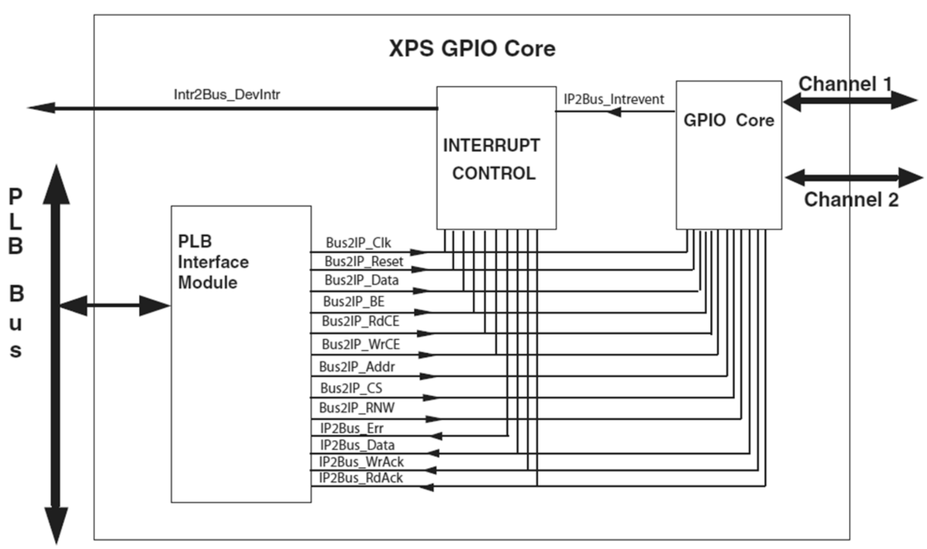



*Figure 1:* **XPS GPIO Block Diagram**

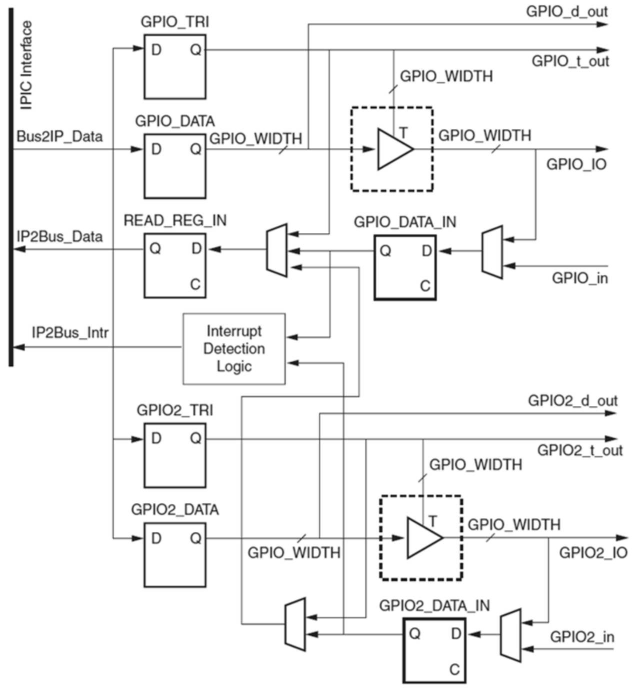



*Table 4:* **XPS GPIO Registers**

| Register Name | Description | PLB Address | Access |
|---|---|---|---|
| GPIO_DATA | Channel 1 XPS GPIO Data Register | C_BASEADDR + 0x00 | Read/Write |
| GPIO_TRI | Channel 1 XPS GPIO 3-state Register | C_BASEADDR + 0x04 | Read/Write |
| GPIO2_DATA | Channel 2 XPS GPIO Data register | C_BASEADDR + 0x08 | Read/Write |
| GPIO2_TRI | Channel 2 XPS GPIO 3-state Register | C_BASEADDR + 0x0C | Read/Write |

*Table 8:* **XPS GPIO Interrupt Registers**

| Register Name | Description | PLB Address | Access |
|---|---|---|---|
| GIE | Global Interrupt Enable Register | C_BASEADDR + 0x11C | Read/Write |
| IP IER | IP Interrupt Enable Register | C_BASEADDR + 0x128 | Read/Write |
| IP ISR | IP Interrupt Status Register | C_BASEADDR + 0x120 | Read/TOW[1] |

# Interrupt Controller - INTC

- Coordinates interrupt requests from modules
- Can handle up to 32 interrupt sources
- Can cascade if necessary for more than 32
- Controlled with programmed I/O techniques
- Can detect edges or levels

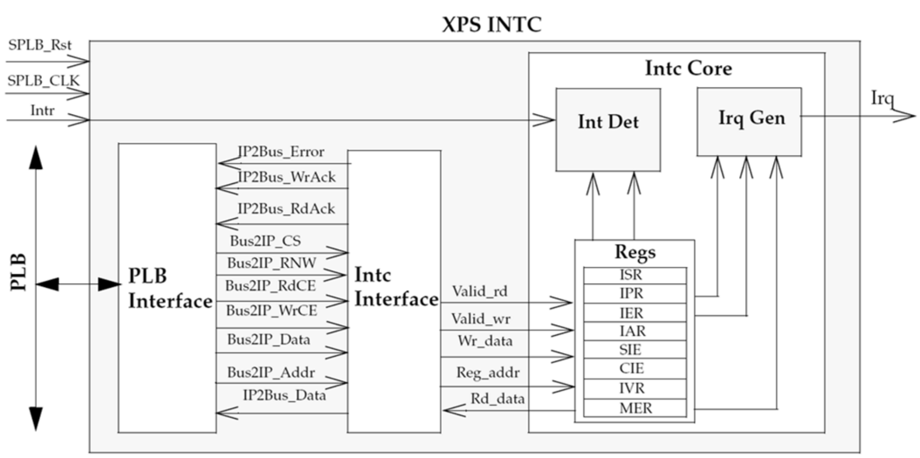


Table 4: **XPS INTC Registers and Base Address Offsets**

| Register Name | Base Address + Offset (Hex) | Access Type | Abbreviation | Reset Value |
|---|---|---|---|---|
| Interrupt Status Register | C_BASEADDR + 0x0 | Read / Write | ISR | All Zeros |
| Interrupt Pending Register | C_BASEADDR + 0x4 | Read only | IPR | All Zeros |
| Interrupt Enable Register | C_BASEADDR + 0x8 | Read / Write | IER | All Zeros |
| Interrupt Acknowledge Register | C_BASEADDR + 0xC | Write only | IAR | All Zeros |
| Set Interrupt Enable Bits | C_BASEADDR + 0x10 | Write only | SIE | All Zeros |
| Clear Interrupt Enable Bits | C_BASEADDR + 0x14 | Write only | CIE | All Zeros |
| Interrupt Vector Register | C_BASEADDR + 0x18 | Read only | IVR | All Ones |
| Master Enable Register | C_BASEADDR + 0x1C | Read / Write | MER | All Zeros |

# PIT – Programmable Interval Timer

- User specifies period for activity by placing count in register
- Counter loaded from register – decrements to zero
- Interrupt can occur when count reaches zero
- Control of PIT: three registers (PIT, TCR, TSR)

## Fixed Interval Timer - FIT

- Four periods to choose from
- Enabled/Disabled with special purpose registers

## Information Representation

- Integer representation – whole numbers
  - Unsigned binary
  - 2's complement
  - Excess codes
- Floating point representation – IEEE 32 bit normalized numbers
- Other: instruction, address, parity, etc.

## Unsigned Binary

$$Value = \sum_{i=0}^{n-1} b_i \times 2^i$$

11110000 = 240

1111000011110000 = 61,680

## Unsigned Binary Patterns

0000 0001 = 1

0000 0010 = 2

0000 0100 = 4

0000 1000 = 8

0000 1010 = 10

0001 0000 = 16

0001 1010 = 16 + 10 = 26

## Unsigned Binary, Fixed Point

$$Value = \sum_{i=0}^{n-1} b_i \times 2^i \times 2^{-p}$$

1111.0000 = 15

11110000.11110000 = 240.9375

1111.000011110000 = 15.05859375

1.111000011110000 = 1.88232421875

## Two's Complement

$$Value = -b_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i$$

11110000 = -16

1111000011110000 = -3,856

## Two’s Complement Patterns

0000 0001 = 1

0000 0010 = 2

0000 0100 = 4

0000 1000 = 8

0000 1010 = 10

0001 0000 = 16

0001 1010 = 16 + 10 = 26

## Two’s Complement Patterns

1000 0001 = 1 + -128 = -127

1000 0010 = 2 + -128 = -126

1000 0100 = 4 + -128 = -124

1000 1000 = 8 + -128 = -120

1000 1010 = 10 + -128 = -118

1001 0000 = 16 + -128 = -112

1001 1010 = 16 + 10 + -128 = -102

## Two's Complement, Fixed Point

$$Value = (-b_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i) \times 2^{-p}$$

1111.0000 = -1

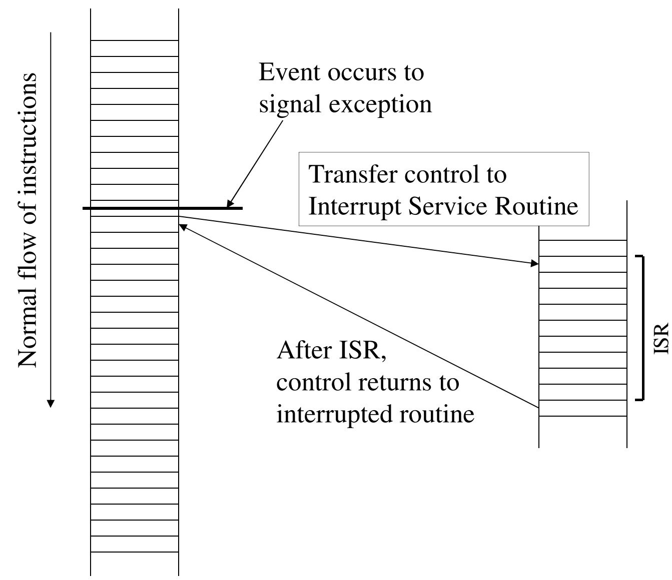
11110000.11110000 = -15.0625

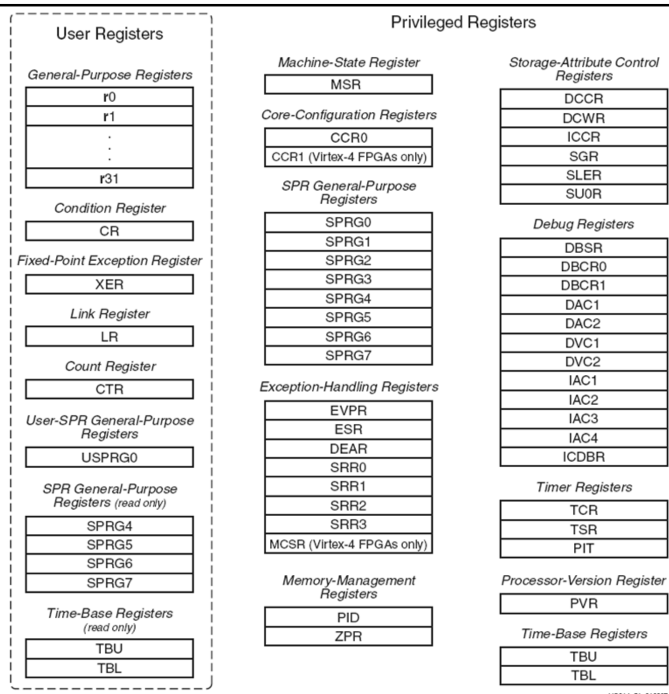
1111.000011110000 = -0.94140625

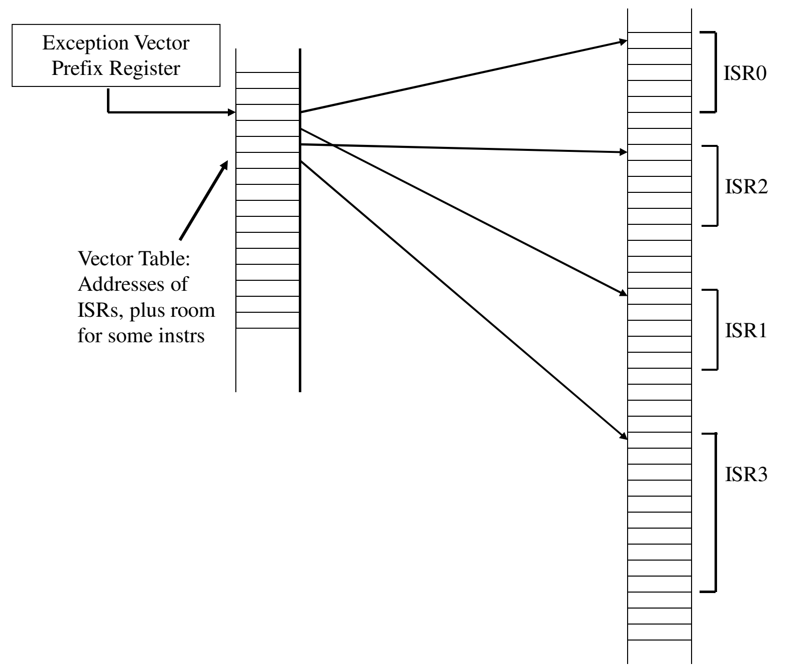
1.111000011110000 = -0.11767578125

## Interrupt System (1)

- Provide timely access to processor facilities
- Mechanism for returning to program without any program change
- Caused by events needing attention
  - Errors (div by 0, illegal instr, priv violation, etc)
  - Intentional events of system (bkpt, traps, etc)
  - External events (timers, UART, GPIO, etc)
- Response activity contained in ISR


Normal flow of instructions
Event occurs to
signal exception
Transfer control to
Interrupt Service Routine
ISR
After ISR,
control returns to
interrupted routine


User Registers
General-Purpose Registers
r0
r1
r31
Condition Register
CR
Fixed-Point Exception Register
XER
Link Register
LR
Count Register
CTR
User-SPR General-Purpose Registers
USPRG0
SPR General-Purpose Registers (read only)
SPRG4
SPRG5
SPRG6
SPRG7
Time-Base Registers (read only)
TBU
TBL
Privileged Registers
Machine-State Register
MSR
Core-Configuration Registers
CCR0
CCR1 (Virtex-4 FPGAs only)
SPR General-Purpose Registers
SPRG0
SPRG1
SPRG2
SPRG3
SPRG4
SPRG5
SPRG6
SPRG7
Exception-Handling Registers
EVPR
ESR
DEAR
SRR0
SRR1
SRR2
SRR3
MCSR (Virtex-4 FPGAs only)
Memory-Management Registers
PID
ZPR
Storage-Attribute Control Registers
DCCR
DCWR
ICCR
SGR
SLER
SU0R
Debug Registers
DBSR
DBCR0
DBCR1
DAC1
DAC2
DVC1
DVC2
IAC1
IAC2
IAC3
IAC4
ICDBR
Timer Registers
TCR
TSR
PIT
Processor-Version Register
PVR
Time-Base Registers
TBU
TBL




# Interrupt System (2)

- Determine cause of interrupt, which ISR to invoke
  - Concept of vector for one of several ISRs
    - difference between PPC and other processors
  - Concept of polling for multiple interrupts that share a single ISR
  - Interrupt system determines which ISR to invoke; ISR looks at appropriate register for polling as needed

# Interrupt System (3): Initialization

- Activities that occur only once, to allow participation in interrupt system or identify details of interrupt process
  - Creation of ISRs to deal with interrupt action
  - Setup of registers in individual functional units
  - Building Int Table, setup of EVPR
  - Setup of MSR (EE; others as needed)

# Machine State Register: Defines Processor State

- Bit 13 - WE - Wait State Enable
- Bit 14 - CE - Critical Interrupt Enable
- Bit 16 - EE - External Interrupt Enable
- Bit 17 - PR - Privilege Level
- Bit 19 - ME - Machine Check Enable
- Bit 21 - DWE - Debug Wait Enable
- Bit 22 - DE - Debug Interrupt Enable
- Bit 26 - IR - Instruction Relocate
- Bit 27 - DR - Data Relocate

## Interrupt System (4): Steady State

- Activity defined by ISR (user supplied)
- In PowerPC, interrupt controller to address offset 0x500
  - Other addresses as appropriate
  - User must deal with possible events efficiently
  - User resets flags, re-establish setup condition
- Return from interrupt – return PC, MSR

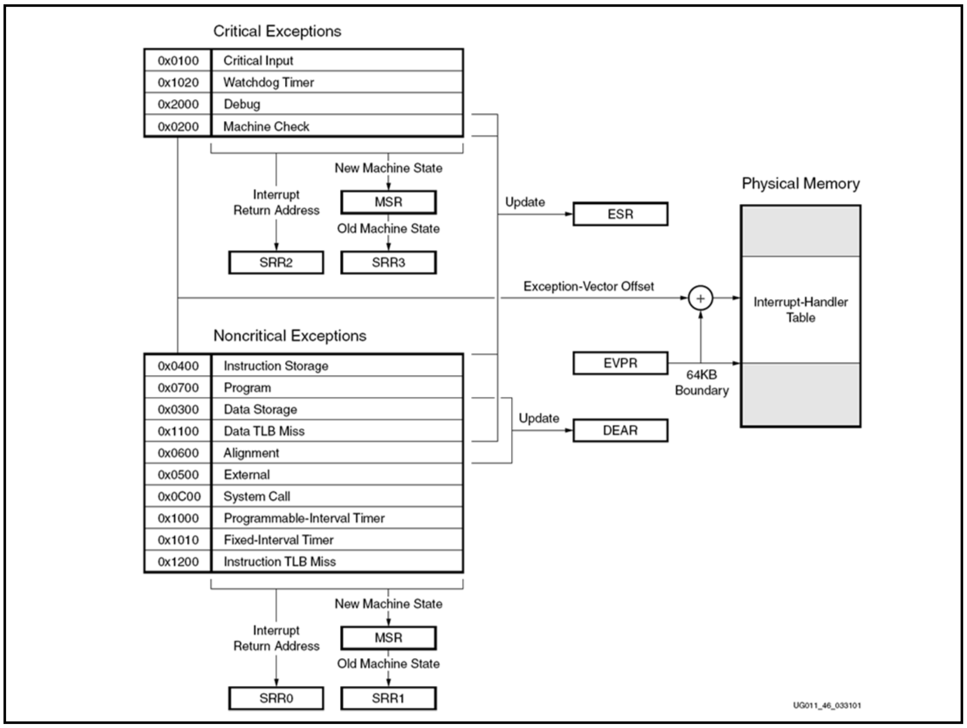
## Machine Characteristic: Precise vs Imprecise

- Pipelining: overlap of operations
- Completion of tasks in order vs out of order
- In order: precise, out of order: imprecise

# Another classification: Critical vs Noncritical

- Processor responds to Critical Exceptions before Noncritical Exceptions
- Critical Exceptions
  - Critical input exception (external)
  - Machine check exception
  - Watchdog timer exception
  - Debug exception
- All other exceptions (PPC405) non-critical

| Exception | Vector Offset | Classification | | | Cause |
|---|---|---|---|---|---|
| Critical Input | 0x0100 | Critical | Asynchronous | Precise | External critical-interrupt signal. |
| Machine Check | 0x0200 | Critical | Asynchronous | Imprecise | External bus error. |
| Data Storage | 0x0300 | Noncritical | Synchronous | Precise | Data-access violation. |
| Instruction Storage | 0x0400 | Noncritical | Synchronous | Precise | Instruction-access violation. |
| External | 0x0500 | Noncritical | Asynchronous | Precise | External noncritical-interrupt signal. |
| Alignment | 0x0600 | Noncritical | Synchronous | Precise | Unaligned operand of **dcread**, **lwarx**, **stwcx.** **dcbz** to non-cacheable or write-through memory. |
| Program | 0x0700 | Noncritical | Synchronous | Precise | Improper or illegal instruction execution. Execution of trap instructions. |
| FPU Unavailable | 0x0800 | Noncritical | Synchronous | Precise | Attempt to execute an FPU instruction when FPU is disabled. |
| System Call | 0x0C00 | Noncritical | Synchronous | Precise | Execution of **sc** instruction. |
| APU Unavailable | 0x0F20 | Noncritical | Synchronous | Precise | Attempt to execute an APU instruction when APU is disabled. |
| Programmable-Interval Timer | 0x1000 | Noncritical | Asynchronous | Precise | Time-out on the programmable-interval timer. |
| Fixed-Interval Timer | 0x1010 | Noncritical | Asynchronous | Precise | Time-out on the fixed-interval timer. |
| Watchdog Timer | 0x1020 | Critical | Asynchronous | Precise | Time-out on the watchdog timer. |
| Data TLB Miss | 0x1100 | Noncritical | Synchronous | Precise | No data-page translation found. |
| Instruction TLB Miss | 0x1200 | Noncritical | Synchronous | Precise | No instruction-page translation found. |
| Debug | 0x2000 | Critical | Asynchronous and synchronous | Precise | Occurrence of a debug event. |




# Interrupt Activity

- Initialization – prepare system for action
  - Initialize individual module to be able to cause interrupt
  - Initialize Interrupt Controller to allow individual interrupts to reach processor
  - Initialize EVPR to identify ISRs
  - Initialize MSR to permit processor to respond to interrupts

# Interrupt Activity

- Before processor initialization – system initialization (planning, preparing)
- Determine desired activity for each module
- Prepare ISR for each module/submodule
- Prepare Table pointing to ISRs

# Interrupt Activity

- Steady State
  - Interrupt breaks fetch-decode-execute cycle
  - Vector activity selects appropriate ISR
  - ISR must:
    - Determine and carry out appropriate activity
    - Take steps to reset/clear interrupt request flag(s)
    - Correctly restore system to previous state