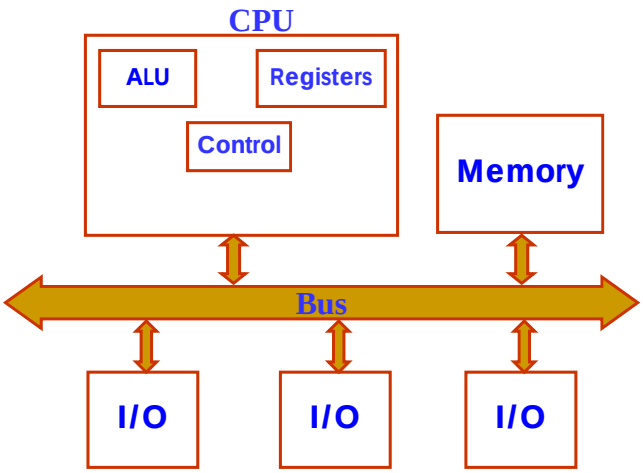
SE 292 (3:0) High Performance Computing
L2: Basic Computer Organization

R. Govindarajan
govind@serc

Basic Computer Organization

- Main parts of a computer system:
  - Processor: Executes programs
  - Main memory: Holds program and data
  - I/O devices: For communication with outside
- Machine instruction: Description of primitive operation that machine hardware is able to execute e.g. ADD these two integers
- Instruction Set: Complete specification of all the kinds of instructions that the processor hardware was built to execute

Basic Computer Organization



3

Inside the Processor…

- Hardware to manage instruction execution
- Arithmetic, logic hardware
- Registers: small units of memory to hold data/instructions temporarily during execution
- Two kinds of registers
  1. Special purpose registers
  2. General purpose registers

4

Special Purpose Registers

- Program Counter (PC): specifies location in memory of instruction being executed
- Instruction Register (IR): holds that instruction
- Processor Status Register: holds status information about current state of processor, such as whether an arithmetic overflow has occurred, etc

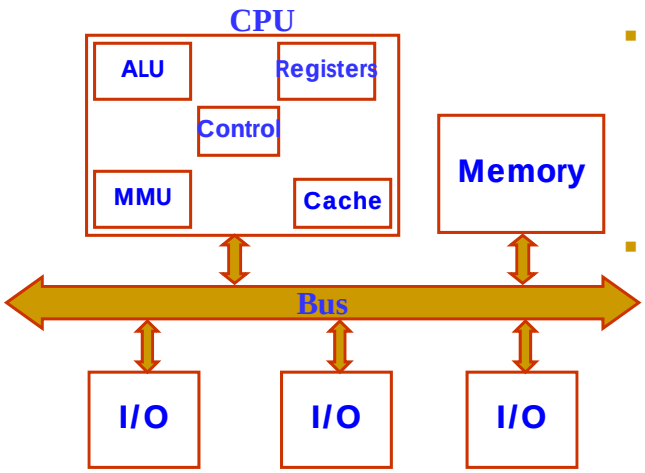
5

General Purpose Registers

- Available for use by programmer, possibly for keeping frequently used data
- Why? Since there is a large speed disparity between processor and main memory
  - 1 GHz Processor: 1 nanosecond time scale
  - Memory: ~ 50 - 100 nsec time scale
- What do these numbers mean?
- Instruction operands can come from registers or from main memory

6

Basic Computer Organization



- **General Purpose Registers**
  - Integer Registers
  - FP Registers
- **Special Purpose Registers**
  - Program Counter
  - Instruction Register

7

Main Memory

- Holds instructions and data
- View as sequence of locations, each referred to by a unique **memory address**
- If size of each memory location is 1 Byte, we call the memory **byte addressable**
- This is quite typical, as smallest data (character) is represented in 1 Byte
- Larger data items are stored in contiguous memory locations, e.g., a 4Byte integer would occupy 4 consecutive memory locations

8

Terms: Byte ordering

In Hexadecimal (0,1,2,…,A,B,C,D,E,F)

Data		1A	C8	B2	46	F0	8C	1E	DF	
Address		400		402		404		406		

What is the integer (4 byte data) at Address 400?

- **Big Endian** byte ordering:**1AC8B246**
  0001 1010 1100 1000 1011 0010 0100 0110 **Decimal: 449,360,454**
- **Little Endian** byte ordering: **46B2C81A**
  0100 0110 1011 0010 1100 1000 0001 1010 **Decimal: 1,186,121,754**

Some machines use big endian byte ordering and others use little endian byte ordering

9

Terms: Word Size, Word Alignment

Word Size

Normal size of an integer or pointer

32b (4B) on many machines

Word Alignment

`Integer variable X is not word aligned'

The data item is not located at a word boundary

Word boundaries: addresses 0, 4, 8, 12, …

HW:

Write a C program to Identify whether a machine supports Little Endian or BigEndian

Write a C program to transfer a sequence of 4-byte values from a Little Endian to BigEndian.

10

Instruction Set Architecture (ISA)

View of the computer visible to the programmer (or compiler)

Two kinds of ISAs

1. **Complex Instruction Set Computer (CISC)**
   A single instruction can perform a complex operation involving several actions

2. **Reduced Instruction Set Computer (RISC)**
   Each instruction performs a only simple operation

11

Instruction Set Architecture

- Description of machine from view of the programmer/compiler
  - Example: Intel x86 ISA
- Includes specification of
  1. The different kinds of instructions available (instruction set)
  2. How operands are specified (addressing modes)
  3. What each instruction looks like (instruction format)

12

Kinds of Instructions

1. Arithmetic/logical instructions
   - Add, subtract, multiply, divide, compare (int/fp)
   - Or, and, not, xor
   - Shift (left/right, arithmetic/logical), rotate
2. Data transfer instructions
   - Load (to register from memory)
   - Store (to memory location from register)
   - Move
3. Control transfer instructions
   - Jump, conditional branch, function call, return
4. Other instructions
   - Example: halt

13

Operand Addressing Modes

- Operands to an instruction
  - Source: input value to instruction
  - Destination: where result is to go
- Addressing Mode
  - How the location of operand is specified
- An operand can be either
  - in a memory location
  - in a register

14

Addressing Modes: Operand in Register

1. **Register Direct** Addressing Mode

   Operand is in the specified general purpose register

   Example

   Suppose that the General Purpose Registers are numbered as 0, 1, 2, etc

   ADD R1, R2, R3 / R1 ← R2 + R3

   destination operand — source operands

R1	59
R2	24
R3	35

2. **Immediate** Addressing Mode

   Operand is included in the instruction

   ADD R1, R2, 1 / R1 ← R2 + 1

15

Addressing Modes: Operand in Memory

3. **Register Indirect** Addressing Mode

   Memory address of operand is in the specified general purpose register

   ADD R1, R1, (R2)

R1	42
R2	100

Address	96	100	104	108
Value	0	10	35	-17

MAIN MEMORY

4. **Base-Displacement** Addressing Mode

   Memory address of operand is calculated as the sum of value in specified register and specified displacement

   ADD R1, R1, 4(R2)

R1	67
R2	100

Address	96	100	104	108
Value	0	10	35	-17

MAIN MEMORY

16

Addressing Modes: Operand in Memory

5. Absolute Addressing Mode

   Memory address of operand is specified directly in the instruction

   ADD R1, R2, #100

6. Indexed Addressing Mode

   Memory address of operand is calculated as sum of contents of 2 registers

   ADD R1, R2, (R3+R4)

- Others
  - Auto-increment/decrement (pre/post)
  - PC relative

17

Case Study: MIPS I Integer Instruction Set

- Registers
  - 32 32b general purpose registers, R0..R31
    - R0 hardwired to value 0
    - R31 implicitly used by instructions JAL, JALR
  - HI, LO: 2 other 32b registers
    - Used implicitly by multiply and divide instructions
- Addressing Modes
  - Immediate, Register direct (arithmetic)
  - Absolute (jumps)
  - Base-displacement (loads, stores)
  - PC relative (branches)

18

MIPS I ISA: General Comments

- All instructions, registers are 32b in size
- Load-store architecture: the only instructions that have memory operands are loads&stores
- Terminology
  - Word: 32b
  - Halfword: 16b
  - Byte: 8b
- Displacements and immediates are signed 16 bit quantities

19

A RISC Instruction Set

Instruction	Mnemonic	Example	Meaning
Data Transfer Instructions			
Load	LB, LBU, LH, LHU, LUI, LW	lw R2, 4(R3)	R2←Mem[R3+4]
Store	SB, SH, SW	sb R2, -8(R4)	Mem[R4 - 8] ← R2
Move	MFHI,MFLO,MTHI, MTLO	mfhi R1	R1← HI

20

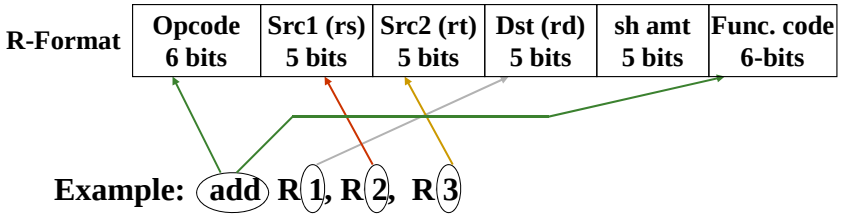
RISC Instruction Set (contd)

Instruction	Mnemonic	Example	Meaning
Control Transfer Instructions			
Conditional Branch	BEQ, BGEZ, BLTZ, BLEZ, BGTZ, BNE	bltz R2, -16	PC←PC+4 –16 if R2 < 0
Jump	J, JR	j <target>	PC←(PC)$_{31-28}$||target||00
Jump & Link	JAL, JALR	jalr R2	R31 ← PC + 8 PC ← R2
System Call	SYSCALL	syscall	

HW:
Write a simple C program and generate the corpg. assembly language program for a RISC/CISC machine. Understand the instructions, function call mechanism, formats of branch and jump instructions, etc.

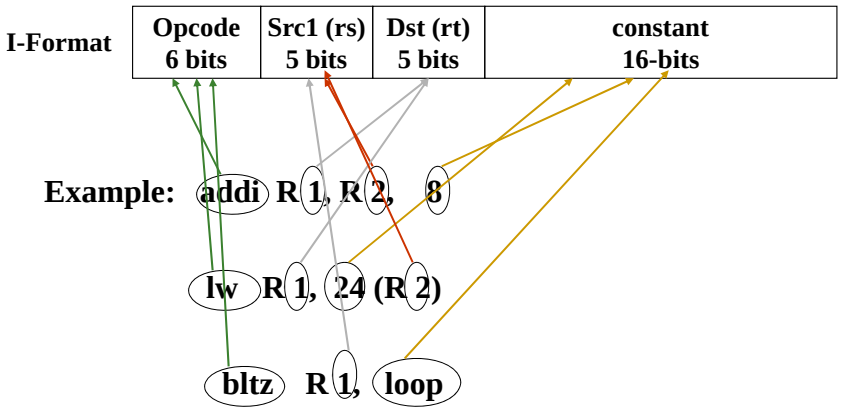
21

MIPS Instruction Encoding



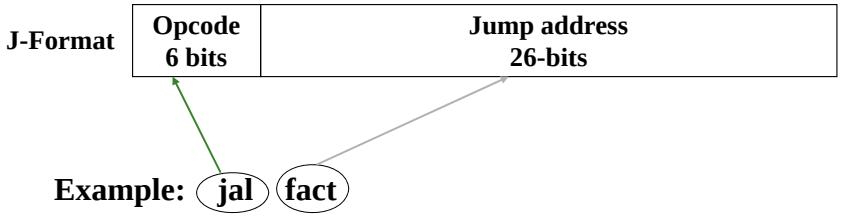
22

MIPS Instruction Encoding



23

MIPS Instruction Encoding



24

CISC vs RISC -- ISA Comparison

a[i++] = a[i] + b[i];
b[i] = b[--i] - 1;

RISC Code:

```
lw  R1, 0(R3)
lw  R2,  0(R4)
add  R5, R1, R2
subi  R2, R2, 1
sw  0(R3), R5
sw   0(R4),  R2
```

CISC Code:

```
add   (R3)+, (R3), (R4)
sub (R4),  -(R4),  1
```

# of Data Memory Accesses:

RISC - 4
CISC - 5

25

On Instruction Processing

- Fetch
  - Get instruction whose address is in PC from memory into IR
  - Increment PC
- Decode
  - Understand instruction, addressing modes, etc
  - Calculate effective addresses and fetch operands
- Execute
  - Do required operation
- Write back the result of the instruction

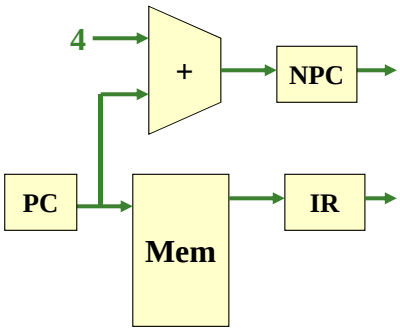
26

Instruction Execution

Instruction Fetch (IF)

from program memory
to instruction register

IR ← Mem [PC]
Increment PC



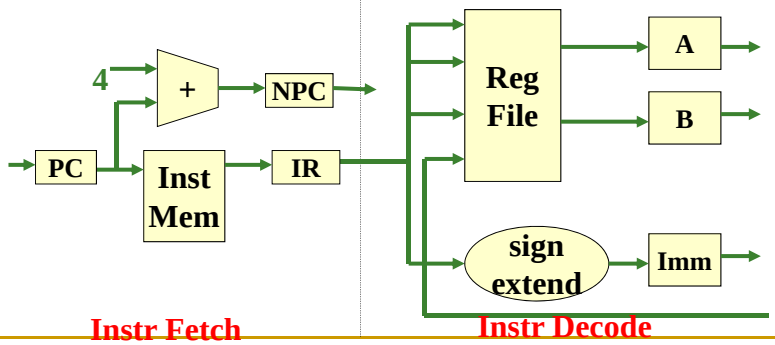
Instr Fetch

27

Instruction Execution…

Instruction Decode & Operand Fetch (ID)

A ← RegisterFile[rs]
B ← RegisterFile[rt]
Imm ← sign extend(IR_{15-0})



Instr Fetch **Instr Decode**

28

Instruction Execution…

Execution (EX)

Arithmetic Inst:

ALU-Out ← A op B

ALU-Out ← A op Imm

Load/Store Inst:

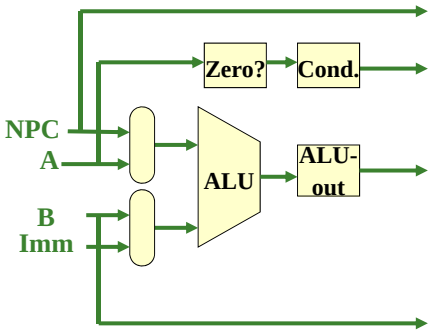
ALU-Out ← A + Imm

Branch Inst:

ALU-Out ← NPC + Imm

Jump Inst:

PC ← NPC $_{31\text{-}28}$ || IR $_{25\text{-}0}$ ||00



Execution

Instruction Execution…

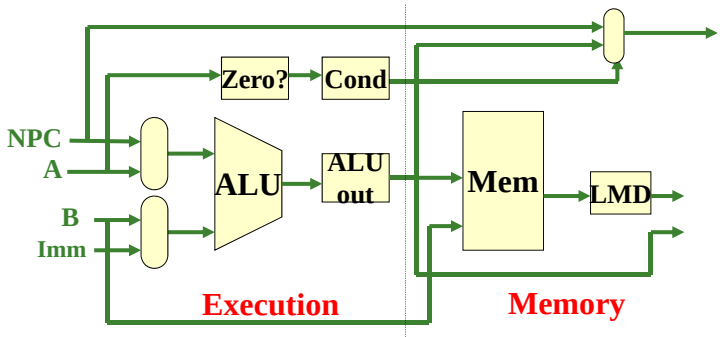
Memory (MEM)

Load Instr

LMD ← Mem[ALUout]

Store Instr

Mem[ALUOut] ← B



Instruction Execution…

Write Back (WB)

ALU Inst

RegisterFile[rd] ← ALUout

Load Inst

RegisterFile[rt] ← LMD

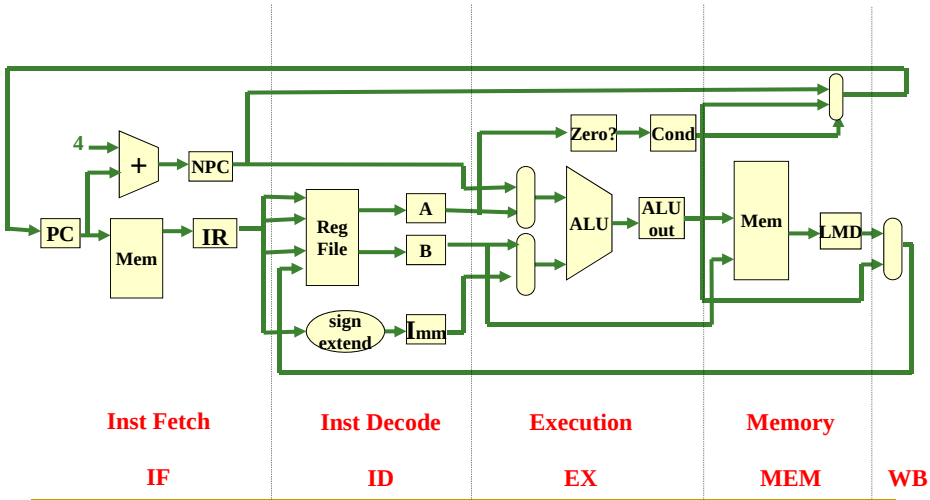
Conditional Branch Inst

PC ← ALU-out if Cond

PC ← NPC otherwise

31

Processor Datapath



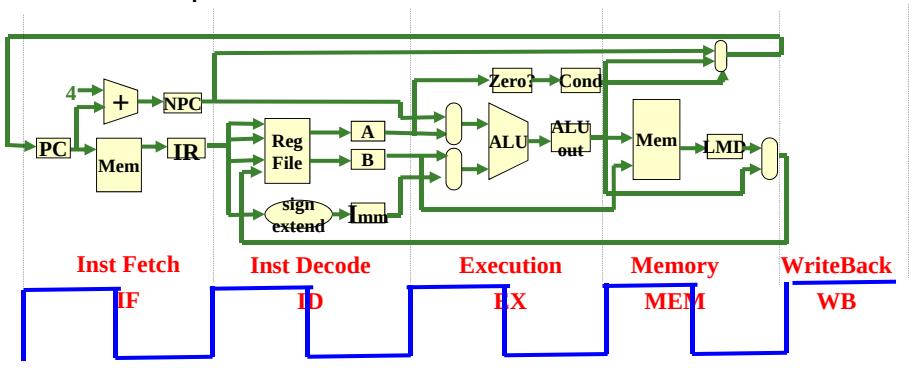
32

Our Assumptions

1. Disparity in Processor vs Memory speed
   - Time for performing addition, register access, etc. vs memory fetch?
   - Which stages require memory access?
2. Main memory delays not typically seen by instruction processor
   - Otherwise timeline is dominated by them
   - There is some hardware mechanism through which most memory access requests can be satisfied at processor speeds (cache memory)
3. Preferable that the time required for each stage of instruction processing to be the same – cycle time

33

- Processor cycle time: time required to do
  - Cache memory access
  - Register access + some logic (like decode)
  - ALU operation



34

Performance of Processor

- Which is more important?
  - execution time of a single instruction
  - throughput of instruction execution
    i.e., number of instructions executed per unit time
- Cycles Per Instruction (CPI)
  - Current ideas: CPI between 3 and 5

35

CPI Calculation

- Cycles for
  - ALU Ins. – 4; Load – 5 ; Store – 4;
    Conditional – 4; Jump – 3;
- % of Instructions in a Program
  - ALU Ins. – 45 %; Load – 15% ; Store – 10% ;
    Conditional – 20% ; Jump – 10%;
- CPI = ?
  - CPI = 0.45*4 + 0.25*5 + 0.1*4 + 0.2*4 + 0.1*3 = 4.55
- How to improve CPI?
  - Pipelining : Fetch the next instruction while the previous is being decoded.

36