CSEN 702

Microprocessors

Lecture 1

Introduction, Review on MIPS64

Reading

Textbook Chapter 1.1, 1.2, 1.3, Appendix A, Appendix C

Part 0 Course content and rules

Personal information

- Email: milad.michel@guc.edu.eg
- Office: C7-209
- Day off: Saturday

- TA’s:
- Sarah Khaled (sarah.khaled@guc.edu.eg)
- Nada Azab (nada.alazab@guc.edu.eg)
- Salma ElShafey (salma.el-shafey@guc.edu.eg)

Course content

Introduction and review of MIPS64
Pipelining review, hazards and performance
Power, energy and performance metrics
Memory hierarchies (1)
Memory hierarchies (2)
Instruction level parallelism, branch prediction
Instruction level parallelism, Dynamic scheduling
VLIW/EPIC
Multiprocessors and Multithreading (1)
Multiprocessors and Multithreading (2)
GPU

- Core course for computer engineers
- Important for research, PHD seekers, and work in R&D in USA and Europe

Textbook: Computer Architecture: A quantitative Approach, 5th Edition, by John L. Hennessy and David A. Patterson, 2012 [ISBN10: 012383872X, ISBN13: 978-8178672663]

Course grading and rules

Assessment	
Student assessment methods	Assessment weighting
Assignments	20%
Project	20%
Midterm Exam	20%
Final Exam	40%

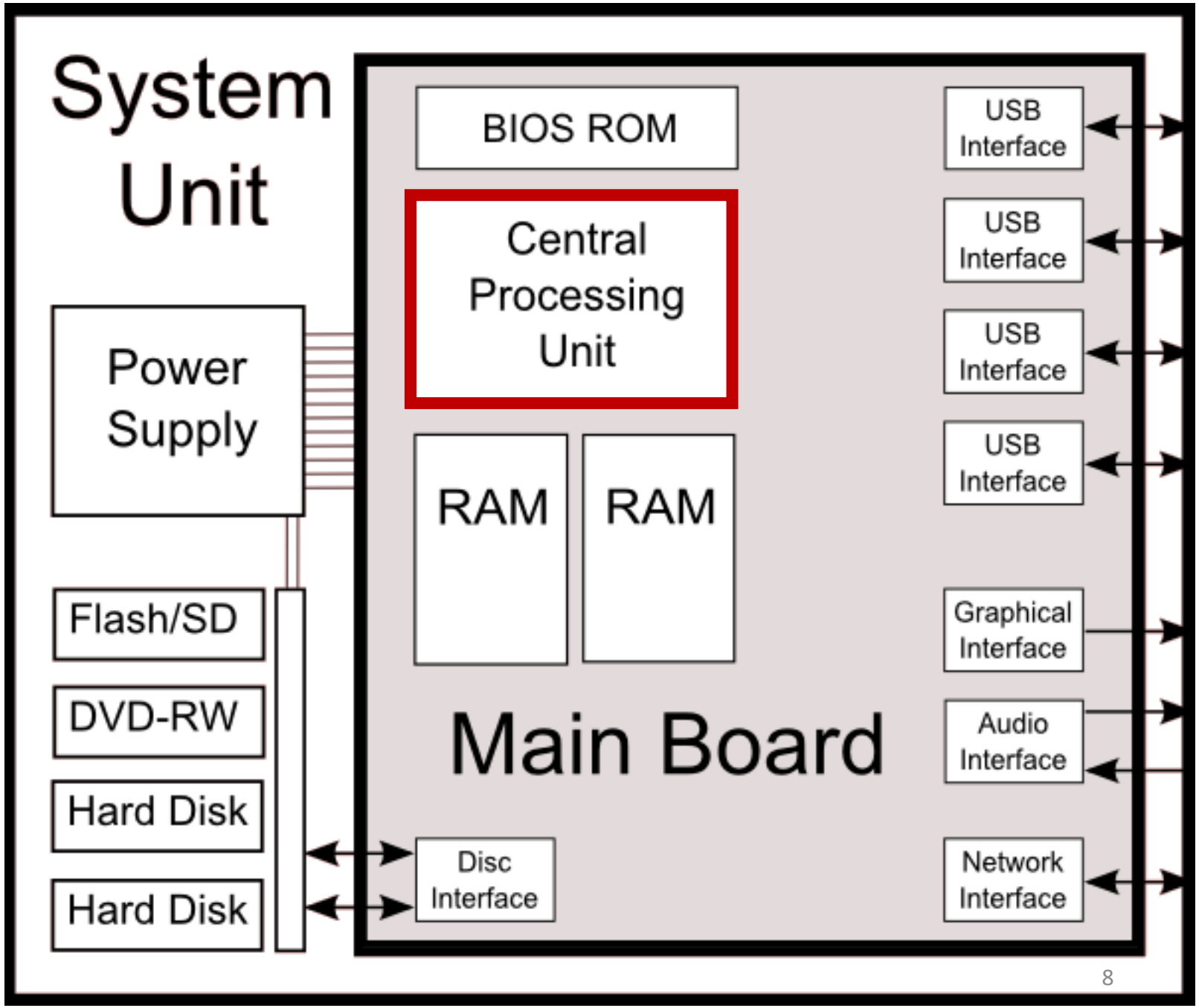
- Bi-weekly course
- Assignments switched for quizzes (best 2 out of 3)
- Attendance in lectures is highly recommended to get the most out of the course
- Schedule and Office hours: Tuesday from 9:30 to 11:30.

contents of the lecture

1. Introduction and history
2. Review of instruction set and MIPS architecture

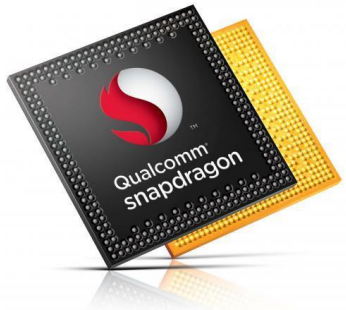
Part 1 Introduction and history

The computer system



History of CPUs

- The brain of the computer
- Microprocessors emerged in late 1970s
- Advances in integrated circuits led to the increased use of microprocessors in business
- The virtual elimination of assembly language coding (thanks to compilers) and vendor-independent operating systems like UNIX led to developing **RISC** architectures.



RISC architectures

- RISC (Reduced Instruction Set Computer) architectures, developed in the early 1980s.
- The RISC-based machines focused the attention of designers on **two** critical performance techniques:

1. The exploitation of **instruction level parallelism** (initially through pipelining and later through multiple instruction issue)

2. The use of **caches** (initially in simple forms and later using more sophisticated organizations and optimizations).

Growth of microprocessor performance since 1970s

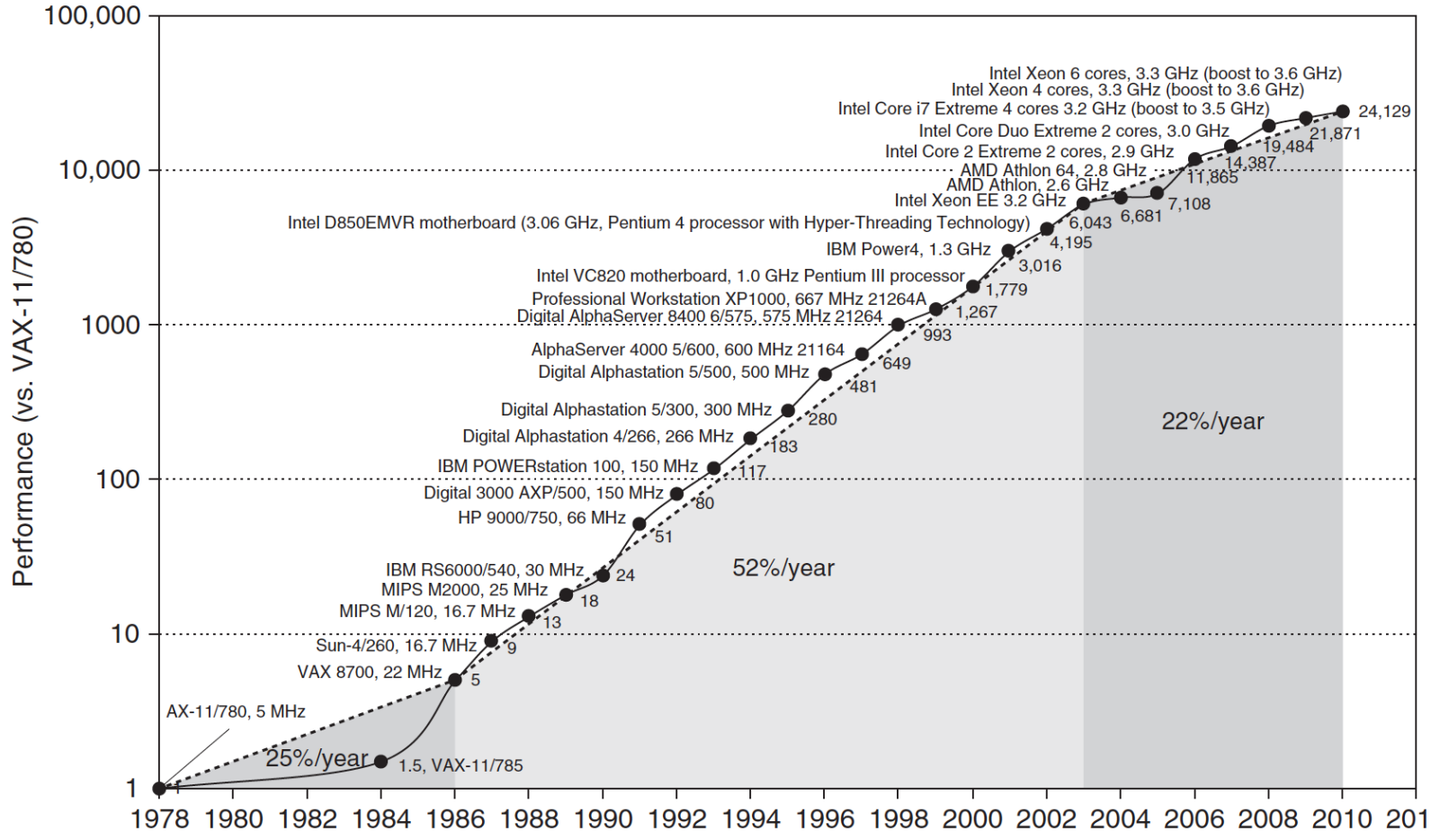
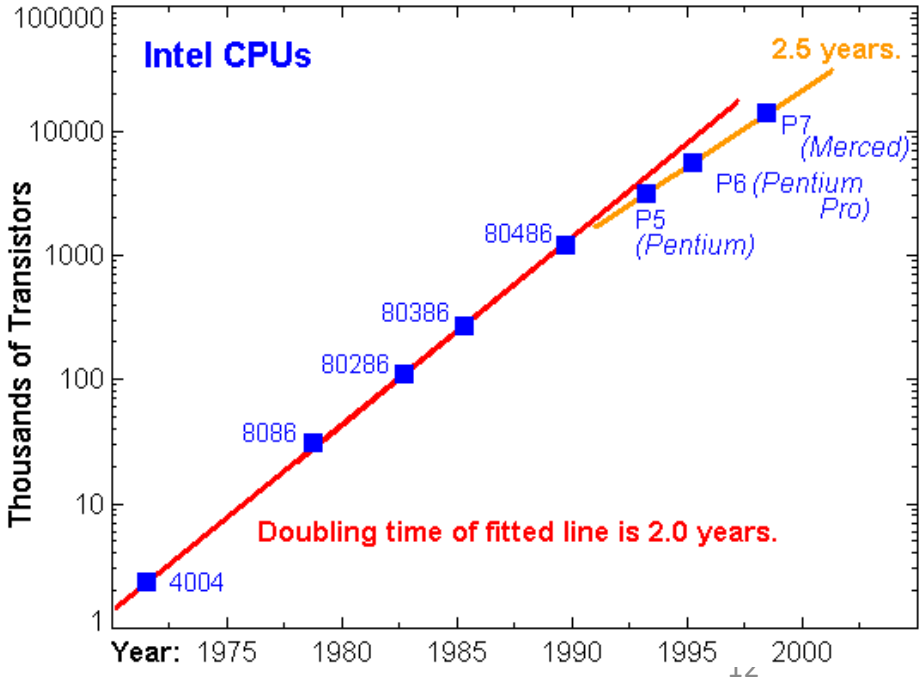


Figure 1.1 Growth in processor performance since the late 1970s. This chart plots performance relative to the VAX

Hardware development

- Improvement in cost-performance leads to new classes of computers: Personal computers and workstations emerged in the 1980s with the availability of the microprocessors

- Continuing improvement of semiconductor manufacturing as predicted by **Moore's*** law has led to the dominance of microprocessor-based computers across the entire range of computer design



***Moore's law** is the observation that the number of transistors in a dense integrated circuit (IC) doubles about every two years

Software development

- This 25,000-fold performance improvement since 1978 allowed programmers today to trade performance for productivity.
- In place of performance-oriented languages like C and C++, much more programming today is done in managed programming languages like **Java** and **C#**.
- Moreover, scripting languages like **Python** and **Ruby**, which are even more productive, are gaining in popularity along with programming frameworks like Ruby on Rails

Uniprocessor vs multiple processors

- In 2004 Intel canceled its high-performance uniprocessor projects and joined others in declaring that the road to higher performance would be via multiple processors per chip rather than via faster uniprocessors.

- Historic **switch** **from** relying solely on *instruction level parallelism* (ILP), **to** *data-level parallelism* (DLP) and *thread-level parallelism* (TLP)

Classes of computers

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Embedded
Price of system	$100–$1000	$300–$2500	$5000–$10,000,000	$100,000–$200,000,000	$10–$100,000
Price of micro-processor	$10–$100	$50–$500	$200–$2000	$50–$250	$0.01–$100
Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance

Figure 1.2 A summary of the five mainstream computing classes and their system characteristics. Sales in 2010

Levels of parallelism

1. **Data-Level Parallelism (DLP)** arises because there are many data items that can be operated on at the same time.

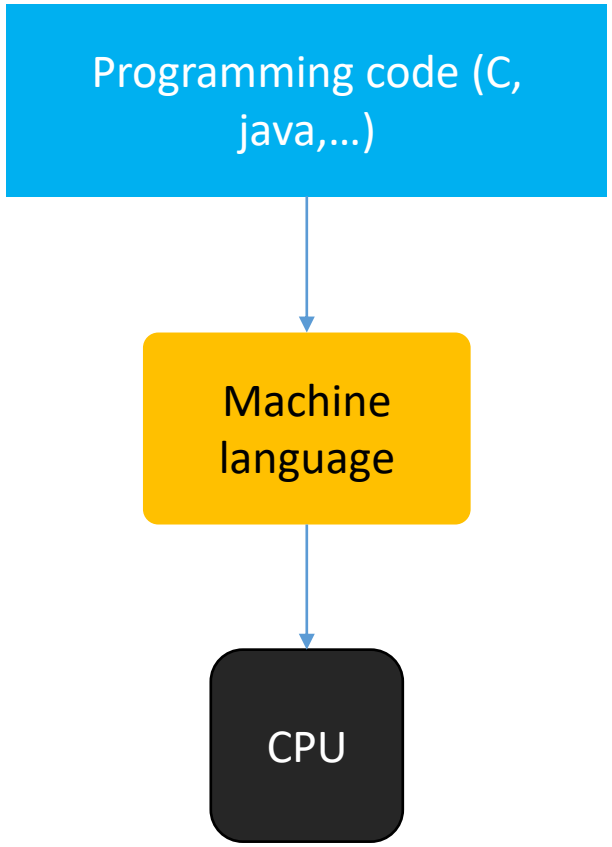
2. **Task-Level Parallelism (TLP)** arises because tasks of work are created that can operate independently and largely in parallel.

Part 2

MIPS 64 instruction set and architecture

Machine language

- Many programming languages are available for users such as java, C, python and so on.
- Before sending them to execution, they must be translated into machine language that the processor understands: **Assembly language** (i.e. MIPS, x86, ArmV8)
- Composed of many instructions that can achieve arithmetic and logic functions
- The entire vocabulary of instructions is called Instruction Set.



How does the instruction set work?

- All operations must be done using **registers** inside the CPU.
- Registers are used to store results or temporary results for later use.
- Data can be fetched (**loaded**) from memory into registers or fetched from registers and **stored** in memory.

Registers in MIPS

register conventions and mnemonics

Number	Name	Use
0	$zero	hardwired 0 value
1	$at	used by assembler (pseudo-instructions)
2-3	$v0-1	subroutine return value
4-7	$a0-3	arguments: subroutine parameter value
8-15	$t0-7	temp: can be used by subroutine without saving
16-23	$s0-7	saved: must be saved and restored by subroutine
24-25	$t8-9	temp
26-27	$k0-1	kernel: interrupt/trap handler
28	$gp	global pointer (static or extern variables)
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	return address for subroutine
	Hi, Lo	used in multiplication (provide 64 bits for result)

Most used

hidden registers

PC, the program counter, which stores the current address of the instruction being executed

IR, which stores the instruction being executed

Quick examples (1)

C

```
int f, g, h, i, j;
f = (g + h) - (i + j);
```

Assume variables are assigned to $s0, $s1, $s2, $s3, $s4 respectively. We will shortly know how to do this.

MIPS

Destination Reg | Source Regs

```
add $s0, $s1, $s2     # $s0 = g + h
add $s1, $s3, $s4     # $s1 = i + j
sub $s0, $s0, $s1     # f = (g + h) - (i + j)
```

Quick examples (2)

C

```
if ( i == j )
  i++ ;
j-- ;
```

Assuming $s1 stores i and $s2 stores j:

MIPS

```
bne $s1, $s2, L1          # branch if !( i == j )
addi $s1, $s1, 1          # i++
L1: addi $s2, $s2, -1     # j--
```

Quick examples (3)

C

```
if ( i == j )
  i++ ;
else
  j-- ;
j += i ;
```

MIPS

```
      bne  $s1, $s2, ELSE   # branch if !( i == j )
      addi $s1, $s1, 1      #    i++
      j NEXT                #    jump over else
ELSE: addi $s2, $s2, -1     # else j--
NEXT:   add $s2, $s2, $s1   # j += i
```

Registers for MIPS64

- MIPS64 has 32 *64-bit* general-purpose registers (GPRs), named R0, R1, . . . , R31. (shown in previous slides)
  - GPRs are also sometimes known as integer registers.

- **Additionally**, there is a set of 32 floating-point registers (FPRs), named F0, F1, . . . , F31, which can hold 32 single-precision SP (32-bit each) values or 32 double-precision DP (64-bit each) values.
  - (When holding one single-precision number, the other half of the FPR is unused.)

- The value of R0 is always 0. (**zero** register)

Data types for MIPS

- The data types are:
  - 8-bit **bytes**, 16-bit **half words**, 32-bit **words,** and 64-bit **double words** for **integer** data
  - 32-bit single precision and 64-bit double precision for **floating point**.

- The MIPS64 operations work on 64-bit integers and 32- or 64-bit floating point.

- Bytes, half words, and words are loaded into the general-purpose registers with either zeros or the sign bit replicated to fill the 64 bits of the GPRs.
  - **Example**:
  - the byte 01111111 is extended to 0000……..0 01111111 because msb is 0
  - the byte 10001111 is extended to 11111111…….10001111 because msb is 1.

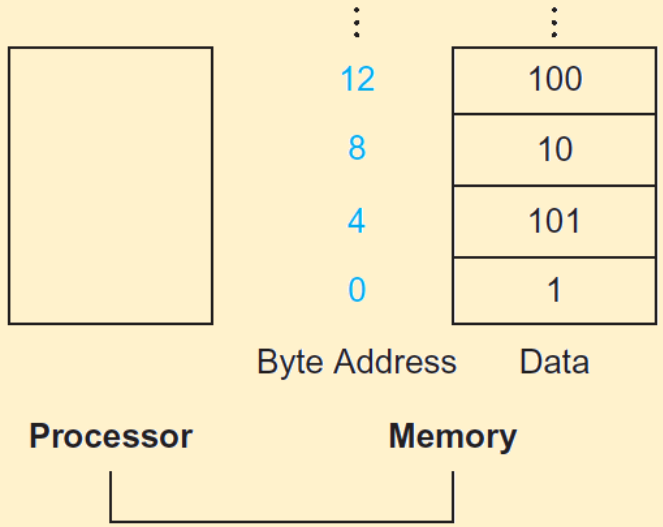
- Once loaded, they are operated on with the 64-bit integer operations

MIPS operations

- There are **four** classes of instructions:
  1. loads and stores
  2. Arithmetic Logic Unit (ALU) operations
  3. branches and jumps (used in loops and functions)
  4. floating-point operations

Reminder about memory:

- Address of data **words** in multiples of 4.
- Each memory location is a byte.



1 load and store

- Any of the general-purpose or floating-point registers may be loaded or stored, except that loading into R0 has no effect.

Instruction type/opcode	Instruction meaning
Data transfers	*Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR*
LB,LBU,SB	Load byte, load byte unsigned, store byte (to/from integer registers)
LH,LHU,SH	Load half word, load half word unsigned, store half word (to/from integer registers)
LW,LWU,SW	Load word, load word unsigned, store word (to/from integer registers)
LD,SD	Load double word, store double word (to/from integer registers)
L.S,L.D,S.S,S.D	Load SP float, load DP float, store SP float, store DP float

Examples on load and store

Example instruction	Instruction name	Meaning
LD R1,30(R2)	Load double word	Regs[R1] $\leftarrow_{64}$ Mem[30+Regs[R2]]
LD R1,1000(R0)	Load double word	Regs[R1] $\leftarrow_{64}$ Mem[1000+0]
LW R1,60(R2)	Load word	Regs[R1] $\leftarrow_{64}$ $(\text{Mem}[60+\text{Regs}[R2]]_0)^{32}$ ## Mem[60+Regs[R2]]
LB R1,40(R3)	Load byte	Regs[R1] $\leftarrow_{64}$ $(\text{Mem}[40+\text{Regs}[R3]]_0)^{56}$ ## Mem[40+Regs[R3]]
LBU R1,40(R3)	Load byte unsigned	Regs[R1] $\leftarrow_{64}$ 0^{56} ## Mem[40+Regs[R3]]
LH R1,40(R3)	Load half word	Regs[R1] $\leftarrow_{64}$ $(\text{Mem}[40+\text{Regs}[R3]]_0)^{48}$ ## Mem[40+Regs[R3]] ## Mem[41+Regs[R3]]

- **Bits are labeled from the most-significant bit starting at 0.**
- The subscript may be a single digit (e.g., Regs[R4]0 yields the sign bit of R4) or a subrange (e.g., Regs[R3]56..63 yields the least-significant byte of R3).

- A superscript is used to replicate a field (e.g., 0^{48} yields a field of zeros of length 48 bits).

- The symbol ## is used to concatenate two fields and may appear on either side of a data transfer.

←n means transfer an n-bit quantity.
We use x, y ← z to indicate that z should be transferred to x and y.

This means that we are loading a 32-bit word from memory at address = [R2] +60, into R1

Adding 60 bytes means 15 words away from address in R2. (since word is 4 bytes)

Since the word is 32 bits, we sign extend using MSB to become 64 bits when storing it in R1.

Examples on load and store

L.S F0,50(R3)	Load FP single	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R3]] ## 0^{32}
L.D F0,50(R2)	Load FP double	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R2]]
SD R3,500(R4)	Store double word	Mem[500+Regs[R4]] $\leftarrow_{64}$ Regs[R3]
SW R3,500(R4)	Store word	Mem[500+Regs[R4]] $\leftarrow_{32}$ Regs[R3]$_{32..63}$
S.S F0,40(R3)	Store FP single	Mem[40+Regs[R3]] $\leftarrow_{32}$ Regs[F0]$_{0..31}$
S.D F0,40(R3)	Store FP double	Mem[40+Regs[R3]] $\leftarrow_{64}$ Regs[F0]
SH R3,502(R2)	Store half	Mem[502+Regs[R2]] $\leftarrow_{16}$ Regs[R3]$_{48..63}$
SB R2,41(R3)	Store byte	Mem[41+Regs[R3]] $\leftarrow_{8}$ Regs[R2]$_{56..63}$

Figure A.23 The load and store instructions in MIPS. All use a single addressing mode and require that the memory value be aligned. Of course, both loads and stores are available for all the data types shown.

This means we are storing whatever value inside register R3, into memory address (500+ values inside R4)

2 Arithmetic and logical

Arithmetic/logical	*Operations on integer or logical data in GPRs; signed arithmetic trap on overflow*
DADD,DADDI,DADDU,DADDIU	Add, add immediate (all immediates are 16 bits); signed and unsigned
DSUB,DSUBU	Subtract; signed and unsigned
DMUL,DMULU,DDIV, DDIVU,MADD	Multiply and divide, signed and unsigned; multiply-add; all operations take and yield 64-bit values
AND,ANDI	And, and immediate
OR,ORI,XOR,XORI	Or, or immediate, exclusive or, exclusive or immediate
LUI	Load upper immediate; loads bits 32 to 47 of register with immediate, then sign-extends
DSLL,DSRL,DSRA,DSLLV, DSRLV,DSRAV	Shifts: both immediate (DS__) and variable form (DS__V); shifts are shift left logical, right logical, right arithmetic
SLT,SLTI,SLTU,SLTIU	Set less than, set less than immediate; signed and unsigned

Instructions starting with D are the same MIPS instructions you know but for 64 bits, we add the D prefix.

Example: ADD for 32 bits, DADD for 64 bits

2 Arithmetic and logical

Examples

Example instruction	Instruction name	Meaning
DADDU R1,R2,R3	Add unsigned	Regs[R1]← Regs[R2]+Regs[R3]
DADDIU R1,R2,#3	Add immediate unsigned	Regs[R1]← Regs[R2]+3
LUI R1,#42	Load upper immediate	Regs[R1]← 0^{32}##42##0^{16}
DSLL R1,R2,#5	Shift left logical	Regs[R1]← Regs[R2]<<5
SLT R1,R2,R3	Set less than	if (Regs[R2]<Regs[R3]) Regs[R1] ←1 else Regs[R1]←0

Figure A.24 Examples of arithmetic/logical instructions on MIPS, both with and without immediates.

- MIPS provides compare instructions, which compare two registers to see if the first is less than the second.
- If the condition is true, these instructions place a 1 in the destination register (to represent true); otherwise, they place the value 0.

3 Control flow instructions

Control	*Conditional branches and jumps; PC-relative or through register*
BEQZ,BNEZ	Branch GPRs equal/not equal to zero; 16-bit offset from PC + 4
BEQ,BNE	Branch GPR equal/not equal; 16-bit offset from PC + 4
BC1T,BC1F	Test comparison bit in the FP status register and branch; 16-bit offset from PC + 4
MOVN,MOVZ	Copy GPR to another GPR if third GPR is negative, zero
J,JR	Jumps: 26-bit offset from PC + 4 (J) or target in register (JR)
JAL,JALR	Jump and link: save PC + 4 in R31, target is PC-relative (JAL) or a register (JALR)

Examples on control

Example instruction		Instruction name	Meaning
J	name	Jump	$PC_{36..63} \leftarrow$ name
JAL	name	Jump and link	Regs[R31]←PC+8; $PC_{36..63}$←name; ((PC + 4)–2^{27}) ≤ name < ((PC+4)+2^{27})
JALR	R2	Jump and link register	Regs[R31]←PC+8; PC←Regs[R2]
JR	R3	Jump register	PC←Regs[R3]
BEQZ	R4,name	Branch equal zero	if (Regs[R4]==0) PC←name; ((PC+4)–2^{17}) ≤ name < ((PC+4)+2^{17})
BNE	R3,R4,name	Branch not equal zero	if (Regs[R3]!= Regs[R4]) PC←name; ((PC+4)–2^{17}) ≤ name < ((PC+4)+2^{17})
MOVZ	R1,R2,R3	Conditional move if zero	if (Regs[R3]==0) Regs[R1]←Regs[R2]

Figure A.25 Typical control flow instructions in MIPS. All control instructions, except jumps to an address in a register, are PC-relative. Note that the branch distances are longer than the address field would suggest; since MIPS instructions are all 32 bits long, the byte branch address is multiplied by 4 to get a longer distance.

If registers R3 and R4 are not equal, jump to label “name”

4 MIPS floating point operations

Floating point	*FP operations on DP and SP formats*
ADD.D,ADD.S,ADD.PS	Add DP, SP numbers, and pairs of SP numbers
SUB.D,SUB.S,SUB.PS	Subtract DP, SP numbers, and pairs of SP numbers
MUL.D,MUL.S,MUL.PS	Multiply DP, SP floating point, and pairs of SP numbers
MADD.D,MADD.S,MADD.PS	Multiply-add DP, SP numbers, and pairs of SP numbers
DIV.D,DIV.S,DIV.PS	Divide DP, SP floating point, and pairs of SP numbers

- The floating-point operations are add, subtract, multiply, and divide;
- A suffix D is used for **double precision**, and a suffix S is used for **single precision** (e.g., ADD.D , ADD.S , SUB.D , SUB.S, MUL.D , MUL.S , DIV.D , DIV.S)

Instruction formats

- We have seen many types of instructions in assembly
- In real implementations, these instructions have to be converted to their binary forms
- All of them are 32-bits long
- But how to differentiate between them?
  - 3 Types of formats: **R-format**, **I-format** and **J-format**

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

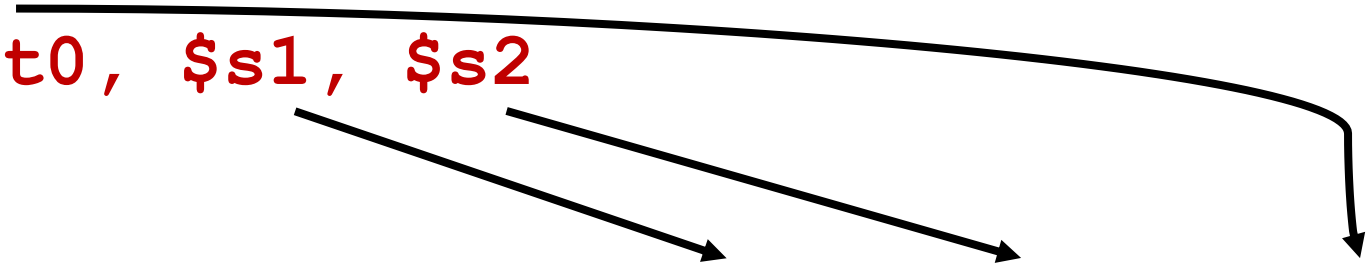
R format instructions

- **op**: Basic operation of the instruction, traditionally called the **opcode**.
- **rs**: The first register source operand.
- **rt**: The second register source operand.
- **rd**: The register destination operand. It gets the result of the operation.
- **shamt**: Shift amount.
- **funct**: Function. This field, often called the function code, selects the specific variant of the operation in the op field.

R-format example

```
add $t0, $s1, $s2
```

Recall this slide

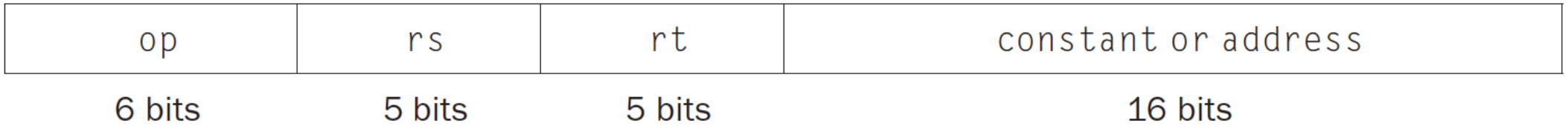


Decimal representation:	0	17	18	8	0	32
Binary:	000000	10001	10010	01000	00000	100000
	6 bits Opcode	5 bits rs	5 bits rt	5 bits rd	5 bits shift	6 bits function

*Opcode 0 is used for almost all ALU operations, the function field is the one to differentiate which function to perform

I format instructions

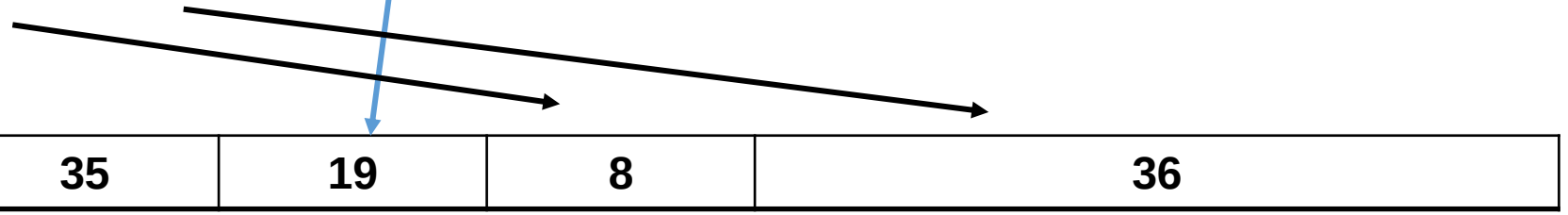
- I-type (for immediate) or I-format and is used by the immediate and data transfer instructions



- The 16-bit address means a load word instruction can load any word within a region of $\pm 2^{15}$ or 32,768 bytes ($\pm 2^{13}$ or 8192 words) of the address in the base register rs.
- Similarly, add immediate is limited to constants no larger than $\pm 2^{15}$.
- The rt field here is the destination

I-format example

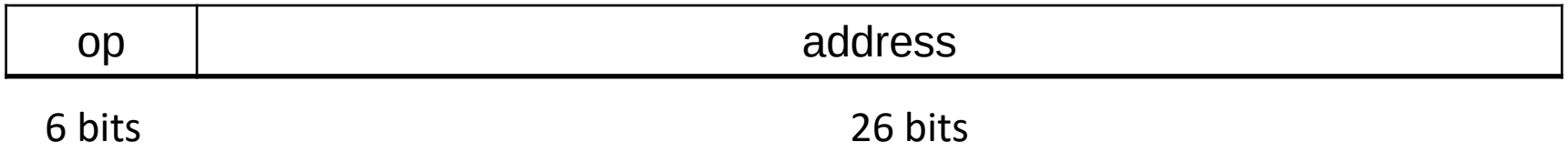
```
lw $t0,36($s3)
```



Decimal	**35**	**19**	**8**	**36**
Binary	**100011**	**10011**	**010000**	**0000000000100100**
	6 bits Opcode	5 bits rs	5 bits rt	16 bits constant or address

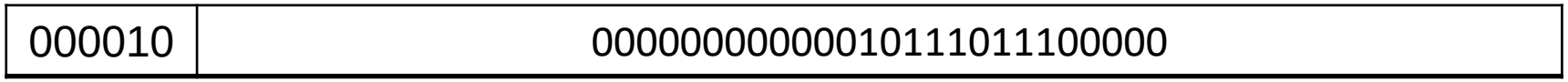
J format instructions

- Used in jump instructions

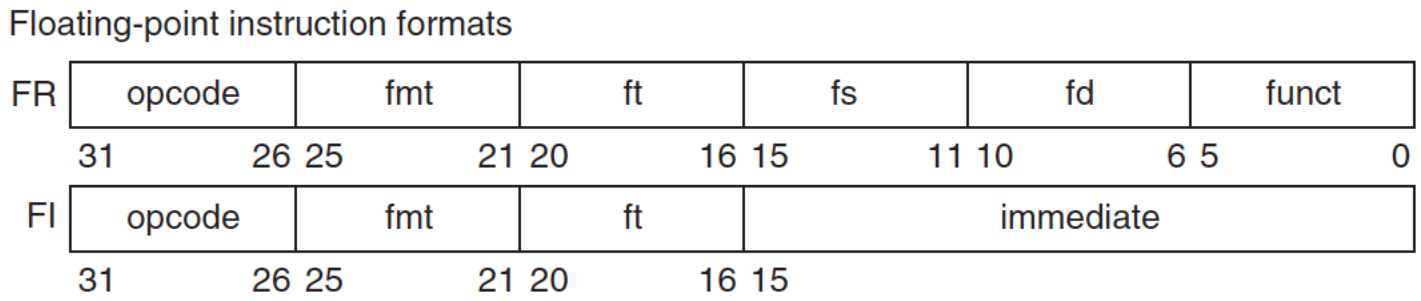


Example

```
J 12000
```



Floating Point formats



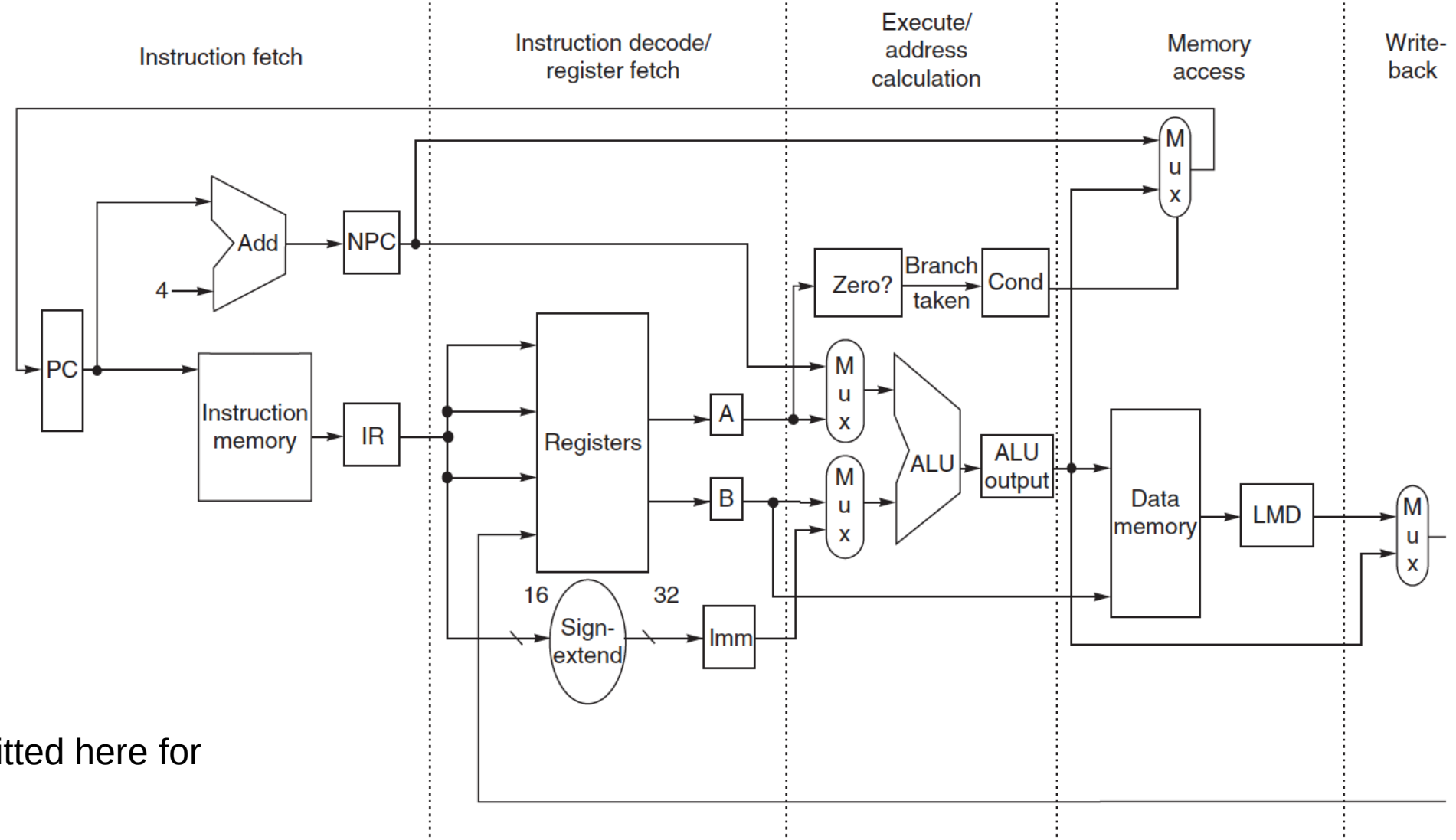
- the **FR** format for floating-point operations, and the **FI** format for floating-point branches.

- the fmt field is used to differentiate between single or double precision.

Single cycle datapath

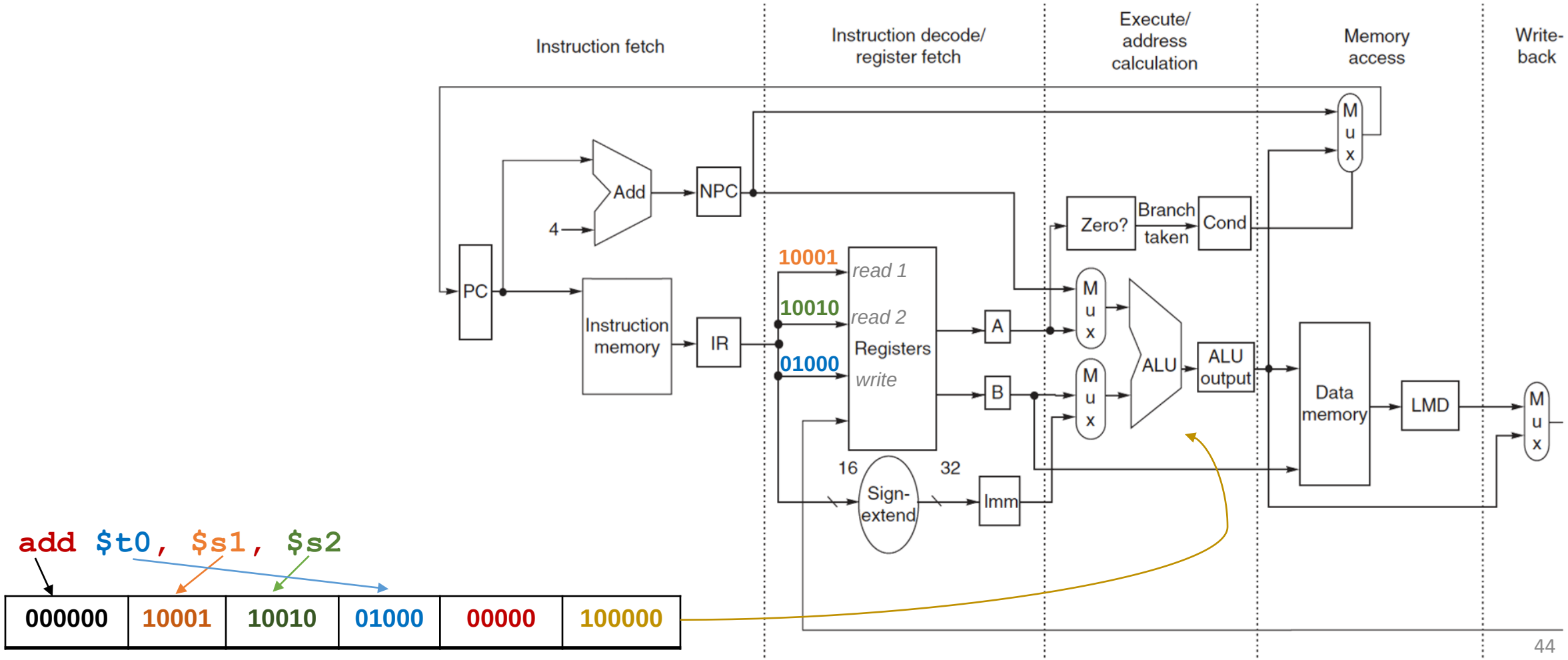
- **Putting all this together: How does the RISC architecture execute instructions?**
- Steps:
  1. **IF**: Fetch the instruction to be executed from memory and update the program counter PC
  2. **ID**: Decode it to see what type of instruction is it and load appropriate registers in the register file.
  3. **EX**: Execute, Arithmetic or logic operations such as additions, address calculations…
  4. **MEM**: memory access to either read or write
  5. **WB**: Write back the results, if needed, to the register file

Single cycle datapath*

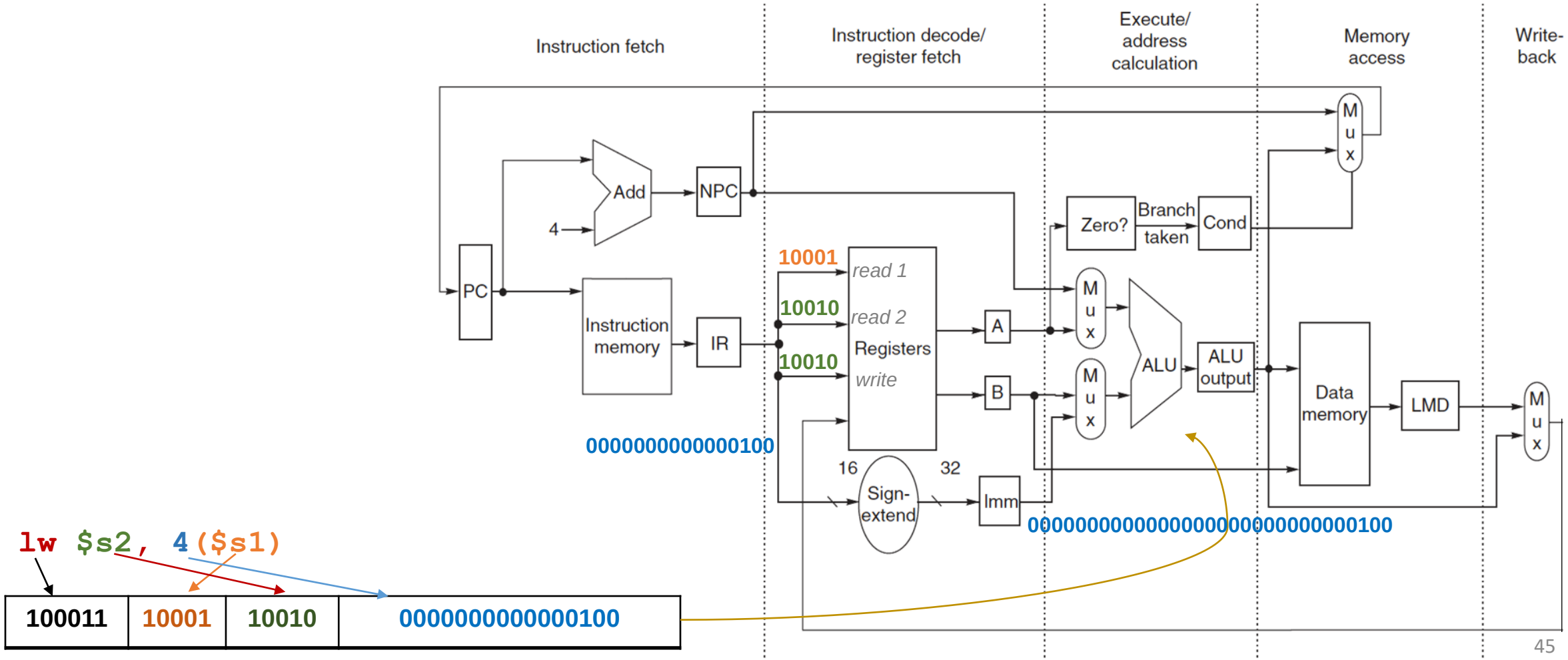


*Many details are omitted here for simplicity

Let's recall how an instruction is executed in the datapath



Let's recall how an instruction is executed in the datapath



Concept of CPI

- **CPI: Clock cycles per instruction**
- **Two** options are available:

  - 1) Execute the instruction in 5 cycles, each cycle is responsible of a stage → short clock cycle
    - Not all instructions execute in the same number of cycles. For example, we can have 60% of the instructions types execute in 3 cycles while the other 40% execute in 4 cycles. What's the CPI?
    - → **Average CPI (clock cycle per instruction) = 0.6 x 3 + 0.4 x 4 = 3.2**

  - 2) Execute it in one cycle (all units are working at the same time) to produce a result → long clock cycle needed to ensure all units finished their job. In this case, CPI = 1

End of lecture 1