EEE304 – Digital Design with HDL (II)

Lecture 5

Dr. Ming Xu

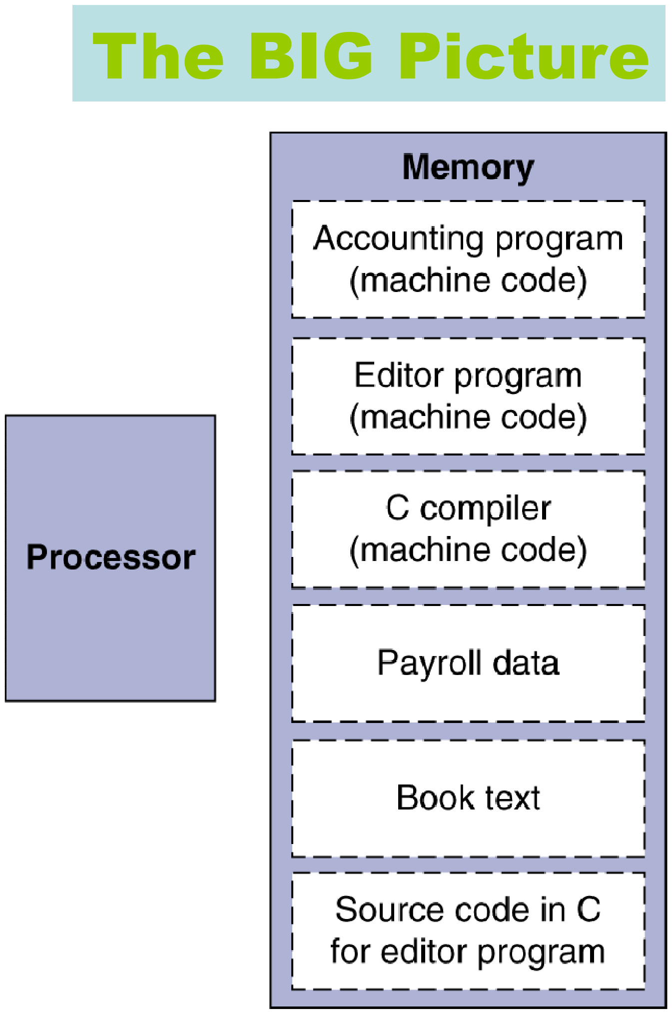
Dept of Electrical & Electronic Engineering

XJTLU

In This Session

- Instructions: Language of the Computer (cont.)

Stored Program Computers



- Instructions represented in binary, just like data
- Instructions and data stored in memory
- Programs can operate on programs
  - e.g., compilers, linkers, …
- Binary compatibility allows compiled programs to work on different computers
  - Standardized ISAs

Conditional Operations

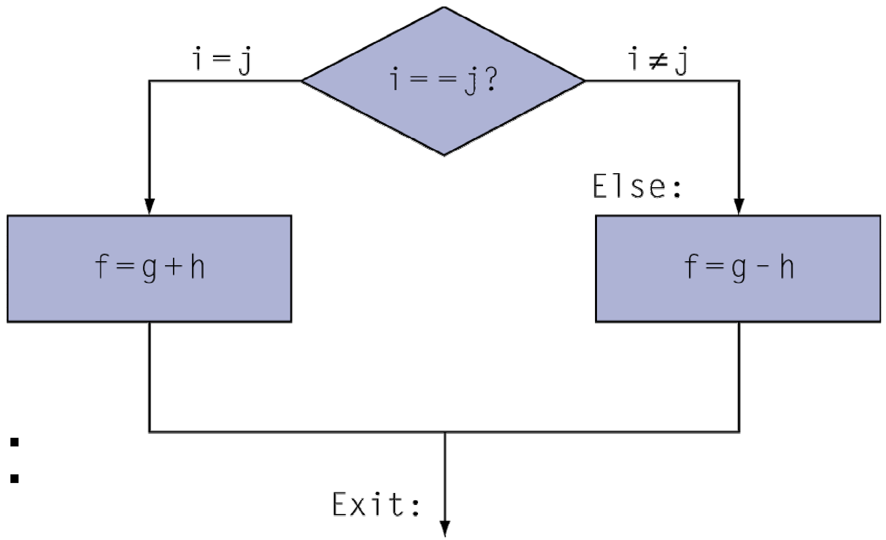
- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially
- `beq rs, rt, L1`
  - if (rs == rt) branch to instruction labeled L1;
- `bne rs, rt, L1`
  - if (rs != rt) branch to instruction labeled L1;
- `j L1`
  - unconditional jump to instruction labeled L1

Compiling If Statements

- C code:

```
if (i==j) f = g+h;
else f = g-h;
```

  – f, g, … in \$s0, \$s1, …



- Compiled MIPS code:

```
       bne $s3, $s4, Else
       add $s0, $s1, $s2
       j   Exit
Else:  sub $s0, $s1, $s2
Exit:  …
```

Assembler calculates addresses

Compiling Loop Statements

- C code:

```
while (save[i] == k) i += 1;
```

  – i in $s3, k in $s5, address of save in $s6

- Compiled MIPS code:

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

More Conditional Operations

- Set result to 1 if a condition is true
  - Otherwise, set to 0
- `slt rd, rs, rt`
  - if (rs < rt) rd = 1; else rd = 0;
- `slti rt, rs, constant`
  - if (rs < constant) rt = 1; else rt = 0;
- Use in combination with `beq`, `bne`

```
slt $t0, $s1, $s2  # if ($s1 < $s2)
bne $t0, $zero, L  #   branch to L
```

Branch Instruction Design

- Why not `blt`, `bge`, etc?
- Hardware for <, ≥, … slower than =, ≠
  - Combining with branch involves more work per instruction, requiring a slower clock
  - All instructions penalized!
- `beq` and `bne` are the common case
- This is a good design compromise

Signed vs. Unsigned

- Signed comparison: `slt`, `slti`
- Unsigned comparison: `sltu`, `sltui`
- Example
  - \$s0 = 1111 1111 1111 1111 1111 1111 1111 1111
  - \$s1 = 0000 0000 0000 0000 0000 0000 0000 0001
  - `slt  $t0, $s0, $s1  # signed`
    - $-1 < +1 \Rightarrow$ \$t0 = 1
  - `sltu $t0, $s0, $s1  # unsigned`
    - $+4,294,967,295 > +1 \Rightarrow$ \$t0 = 0

Branch Addressing

- Branch instructions specify
  - Opcode, two registers, target address
- Most branch targets are near branch
  - Forward or backward

op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

- PC-relative addressing
  - Target address = PC + offset $\times$ 4
  - PC already incremented by 4 by this time

MIPS I-format Instructions

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Immediate arithmetic instructions
  - rs: source register
  - rt: destination register
  - Constant: -2^{15} to $+2^{15} - 1$
- load/store instructions
  - rs: base address
  - rt: destination register (lw) or source register (sw)
  - Address: offset address

Jump Addressing (MIPS J-format Instructions)

- Jump (`j`) targets could be anywhere in text segment
  - Encode full address in instruction

op	address
6 bits	26 bits

- (Pseudo)Direct jump addressing
  - Target address = $PC_{31\ldots28}$: (address $\times$ 4)

Branching Far Away

- If branch target is too far to encode with 16-bit offset, assembler rewrites the code
- Example

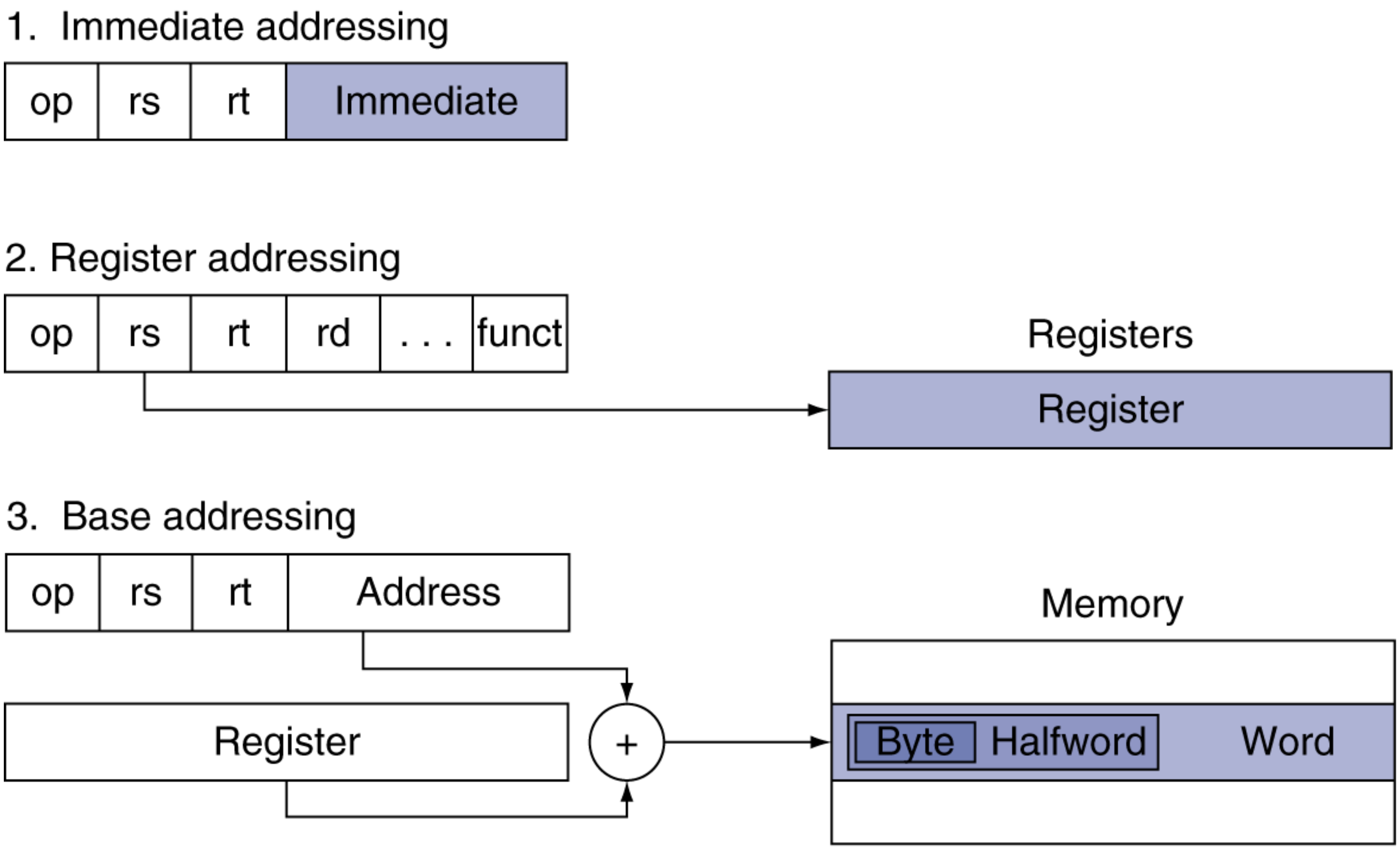
```
      beq $s0,$s1, L1
              ↓
      bne $s0,$s1, L2
      j L1
  L2: …
```

Target Addressing Example

- Loop code from earlier example
  - Assume Loop at location 80000

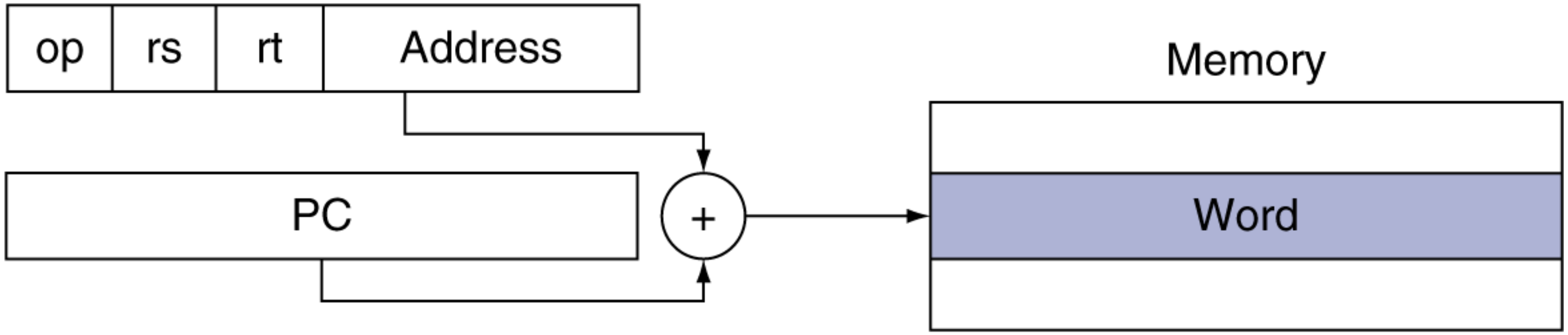
`Loop: sll  $t1, $s3, 2`	80000	0	0	19	9	2	0
`      add  $t1, $t1, $s6`	80004	0	9	22	9	0	32
`      lw   $t0, 0($t1)`	80008	35	9	8	0		
`      bne  $t0, $s5, Exit`	80012	5	8	21	2		
`      addi $s3, $s3, 1`	80016	8	19	19	1		
`      j    Loop`	80020	2	20000				
`Exit: …`	80024						

Addressing Mode Summary



Addressing Mode Summary

4. PC-relative addressing



5. Pseudodirect addressing