

CS2100

<http://www.comp.nus.edu.sg/~cs2100/>

COMPUTER ORGANISATION

Lecture #11

The Processor: Datapath

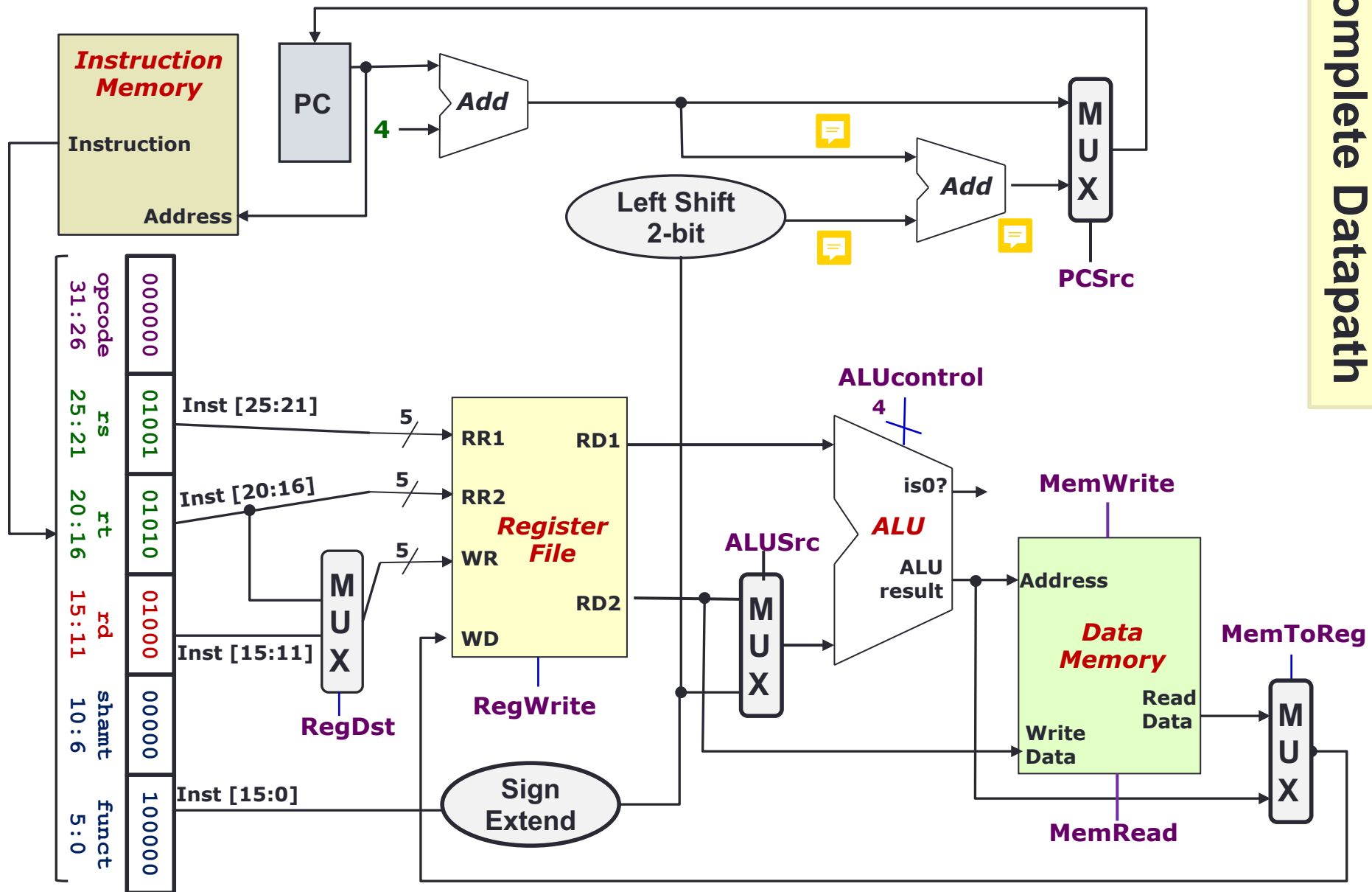


NUS
National University
of Singapore

School of
Computing

Lecture #11a: Processor: Datapath

1. The Complete Datapath!
2. Brief Recap
3. From C to Execution
 - 3.1 Writing C Program
 - 3.2 Compiling to MIPS
 - 3.3 Assembling to Binaries
 - 3.4 Execution (Datapath)



2. Brief Recap

■ Lecture #7, Slide 4

High-level
language
program
(in C)

```
swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Compiler

Assembly
language
program
(for MIPS)

```
swap:
    muli $2, $5, 4
    add $2, $4, $2
    lw $15, 0($2)
    lw $16, 4($2)
    sw $16, 0($2)
    sw $15, 4($2)
    jr $31
```

Assembler

Binary machine
language
program
(for MIPS)

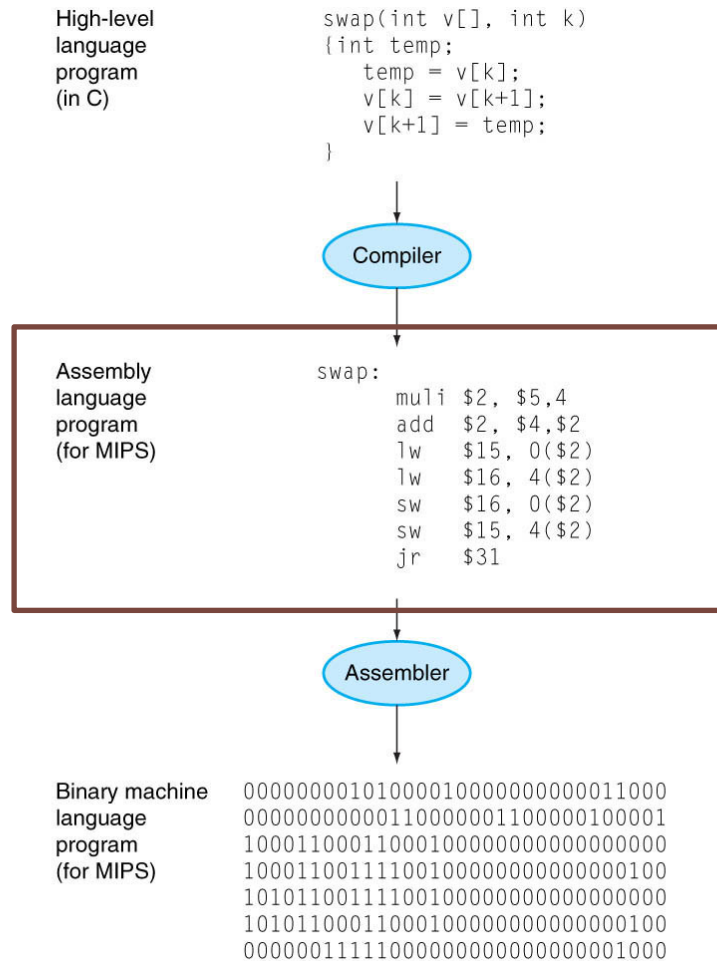
```
000000001010000100000000000011000
000000000000110000001100000100001
1000110001100010000000000000000
10001100111100100000000000000100
1010110011110010000000000000000
10101100011000100000000000000100
0000001111100000000000000001000
```

➤ Write program in high-level language (e.g., **C**)

```
if (x != 0) {
    a[0] = a[1] + x;
}
```

2. Brief Recap

■ Lecture #7, Slide 4



➤ **Compiler** translates to assembly language (e.g., **MIPS**)

```

beq $16, $0, Else
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)
  
```

Else:

2. Brief Recap

■ Lecture #7, Slide 4

High-level
language
program
(in C)

```
swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Compiler

Assembly
language
program
(for MIPS)

```
swap:
    muli $2, $5, 4
    add $2, $4, $2
    lw $15, 0($2)
    lw $16, 4($2)
    sw $16, 0($2)
    sw $15, 4($2)
    jr $31
```

Assembler

Binary machine
language
program
(for MIPS)

```
000000001010000100000000000011000
000000000000110000001100000100001
100011000110001000000000000000000
100011001111001000000000000000100
101011001111001000000000000000000
101011000110001000000000000000100
000000111110000000000000000001000
```

➤ **Assembler** translates to machine code (i.e., **binaries**)

```
0001 0010 0000 0000
0000 0000 0000 0011
```

```
1000 1110 0010 1000
0000 0000 0000 0100
```

```
0000 0010 0000 1000
0100 0000 0001 0100
```

```
1010 1110 0010 1000
0000 0000 0000 0000
```

2. Brief Recap

■ Lecture #7, Slide 4

High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

Compiler

Assembly
language
program
(for MIPS)

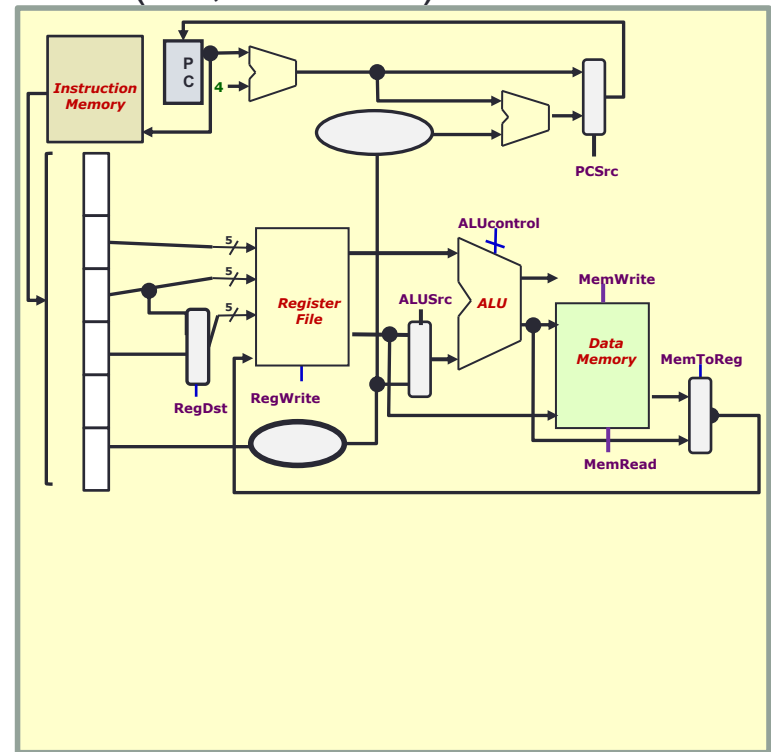
```
swap:
  muli $2, $5, 4
  add $2, $4, $2
  lw $15, 0($2)
  lw $16, 4($2)
  sw $16, 0($2)
  sw $15, 4($2)
  jr $31
```

Assembler

Binary machine
language
program
(for MIPS)

```
00000000101000010000000000011000
000000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
0000001111100000000000000001000
```

➤ **Processor** executes the machine code (i.e., **binaries**)



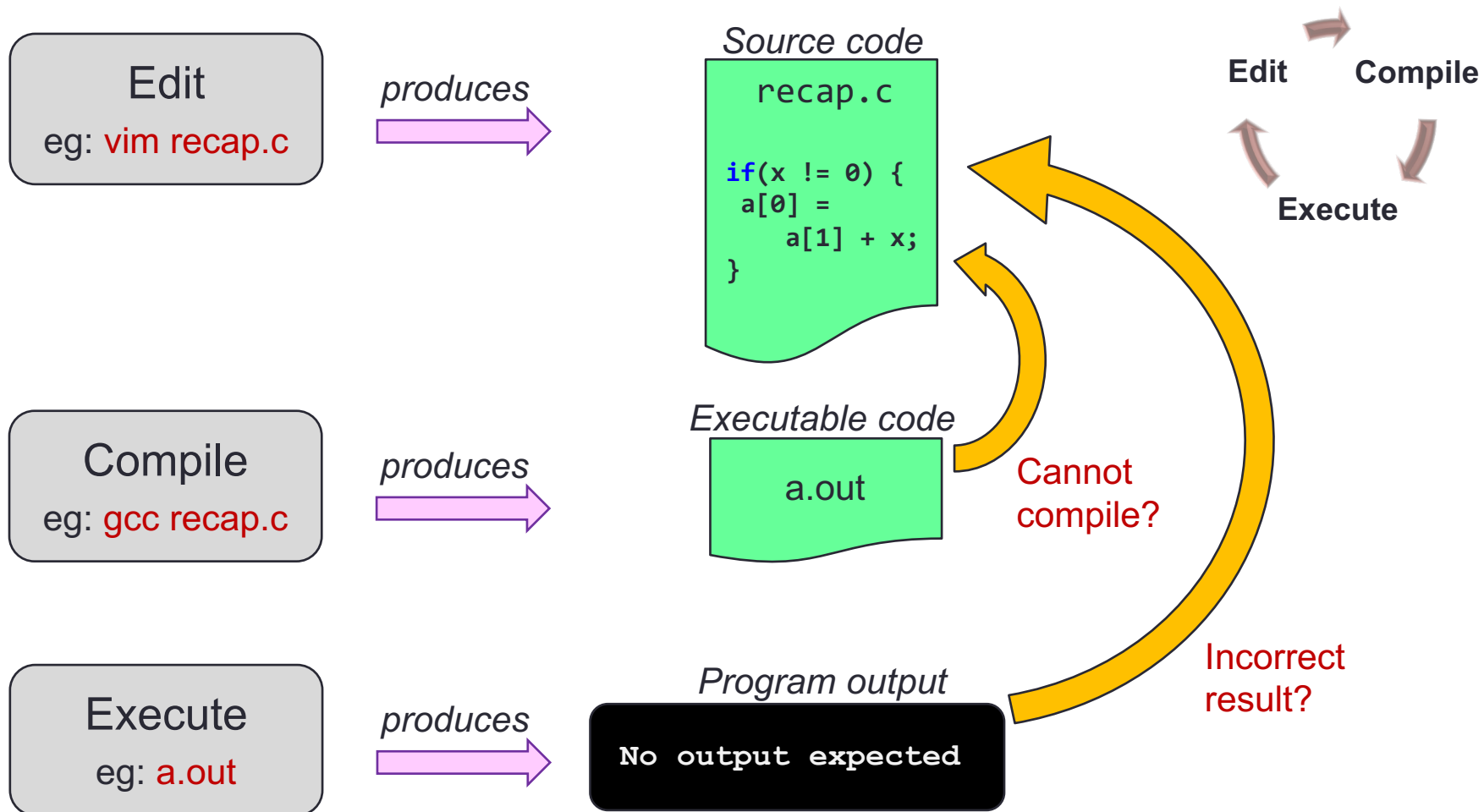
3. From C to Execution

- We play the role of **Programmer**, **Compiler**, **Assembler**, and **Processor**
 - Program:

```
if (x != 0) {  
    a[0] = a[1] + x;  
}
```
 - Programmer:
 - Show the workflow of compiling, assembling, and executing C program
 - Compiler:
 - Show how the program is compiled into MIPS
 - Assembler:
 - Show how the MIPS is translated into binaries
 - Processor:
 - Show how the datapath is activated in the processor

3.1 Writing C Program

- Edit, Compile, Execute: Lecture #2, Slide 5



3.2 Compiling to MIPS

- Key Idea #1:

Compilation is a *structured process*

```
if(x != 0) {  
    a[0] = a[1] + x;  
}
```

- Each structure can be compiled *independently*

Inner Structure

```
a[0] = a[1] + x;
```

Outer Structure

```
if (x != 0) {  
  
  
  
  
  
  
  
  
  
}
```

recap.c

```
if(x != 0) {  
    a[0] =  
        a[1] + x;  
}
```

3.2 Compiling to MIPS

- Key Idea #2:
Variable-to-Register Mapping

```
if(x != 0) {  
    a[0] = a[1] + x;  
}
```

- Let the mapping be:

Variable	Register Name	Register Number
x	\$s0	\$16
a	\$s1	\$17

recap.c

```
if(x != 0) {  
    a[0] =  
        a[1] + x;  
}
```

3.2 Compiling to MIPS

- Common Technique #1:
Invert the condition for shorter code
(Lecture #8, Slide 22)

Mapping:

x: \$16
a: \$17

recap.c

```
if(x != 0) {  
    a[0] =  
        a[1] + x;  
}
```

Outer Structure

```
if(x != 0) {  
  
  
  
}
```

Outer MIPS Code

```
        beq $16, $0, Else  
  
        # Inner Structure  
  
Else:
```

3.2 Compiling to MIPS

- Common Technique #2:
Break complex operations, use temp register
(Lecture #7, Slide 29)

Mapping:

```
x: $16  
a: $17  
$t1: $8
```

recap.c

```
if(x != 0) {  
    a[0] =  
        a[1] + x;  
}
```

Inner Structure

```
a[0] = a[1] + x;
```

Simplified Inner Structure

```
$t1 = a[1];  
$t1 = $t1 + x;  
a[0] = $t1;
```

3.2 Compiling to MIPS

- Common Technique #3:
Array access is **lw**, array update is **sw**
(Lecture #8, Slide 13)

Mapping:

```
x: $16  
a: $17  
$t1: $8
```

recap.c

```
if(x != 0) {  
    a[0] =  
        a[1] + x;  
}
```

Simplified Inner Structure

```
$t1 = a[1];  
$t1 = $t1 + x;  
a[0] = $t1;
```

Inner MIPS Code

```
lw    $8, 4($17)  
add   $8, $8, $16  
sw    $8, 0($17)
```

3.2 Compiling to MIPS

- Common Error #1:

Assume that the address of the next word can be found by incrementing the address in a register by 1 instead of by the word size in bytes

- Example:

- `$t1 = a[1];`
is translated to
`lw $8, 4($17)`
instead of
`lw $8, 1($17)`

Mapping:

```
x: $16  
a: $17  
$t1: $8
```

recap.c

```
if(x != 0) {  
    a[0] =  
        a[1] + x;  
}
```

3.2 Compiling to MIPS

- Last Step:

Combine the two structures logically

Mapping:

```
x: $16
a: $17
$t1: $8
```

recap.c

```
if(x != 0) {
    a[0] =
        a[1] + x;
}
```

Inner MIPS Code

```
lw    $8, 4($17)
add   $8, $8, $16
sw    $8, 0($17)
```

Outer MIPS Code

```
beq $16, $0, Else
```

```
# Inner Structure
```

```
Else:
```

Combined MIPS Code

```
beq $16, $0, Else
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)
```

```
Else:
```


recap.mips

```

beq $16, $0, Else
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)

```

Else:

3.3 Assembling to Binary

■ Instruction Types Used:

1. R-Format: (Lecture #9, Slide 8)

- opcode $\$rd$, $\$rs$, $\$rt$



2. I-Format: (Lecture #9, Slide 14)

- opcode $\$rt$, $\$rs$, immediate



3. Branch: (Lecture #9, Slide 22)

- Uses I-Format
- $PC = (PC + 4) + (immediate \times 4)$

recap.mips

```

beq $16, $0, Else
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)

```

Else:

3.3 Assembling to Binary

- **beq \$16, \$0, Else**
 - Compute immediate value (Lecture #9, Slide 27)
 - **immediate** = 3
 - Fill in fields (*refer to MIPS Reference Data*)

6	5	5	16
4	16	0	3

- Convert to binary

000100	10000	00000	000000000000000011
--------	-------	-------	--------------------

```
beq $16, $0, Else
```

```
lw  $8, 4($17)
```

```
add $8, $8, $16
```

```
sw  $8, 0($17)
```

+3

Else: ←

recap.mips

```

beq $16, $0, Else
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)

```

Else:

3.3 Assembling to Binary

■ `lw $8, 4($17)`

■ Fill in fields (*refer to MIPS Reference Data*)

6	5	5	16
35	17	8	4

■ Convert to binary

100011	10001	01000	00000000000000100
--------	-------	-------	-------------------

0001 0010 0000 0000 0000 0000 0000 0011

lw \$8, 4(\$17)

add \$8, \$8, \$16

sw \$8, 0(\$17)

Else:

3.3 Assembling to Binary

■ **add \$8, \$8, \$16**

■ Fill in fields (*refer to MIPS Reference Data*)

6	5	5	5	5	6
0	8	16	8	0	32

■ Convert to binary

000000	01000	10000	01000	00000	100000
--------	-------	-------	-------	-------	--------

recap.mips

```
beq $16, $0, Else
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)
```

Else:

```
0001 0010 0000 0000 0000 0000 0000 0011
1000 1110 0010 1000 0000 0000 0000 0100
```

```
    add $8, $8, $16
```

```
    sw  $8, 0($17)
```

Else:

recap.mips

```

beq $16, $0, Else
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)

```

Else:

3.3 Assembling to Binary

■ **sw** **\$8**, **0** (**\$17**)

■ Fill in fields (*refer to MIPS Reference Data*)

6	5	5	16
43	17	8	0

■ Convert to binary

101011	10001	01000	0000000000000000
--------	-------	-------	------------------

```

0001 0010 0000 0000 0000 0000 0000 0011
1000 1110 0010 1000 0000 0000 0000 0100
0000 0001 0001 0000 0100 0000 0010 0000

```

sw **\$8**, **0** (**\$17**)

Else:

3.3 Assembling to Binary

- Final Binary
 - Hard to read?
 - Don't worry, this is intended for machine not for human!

recap.mips

```
beq $16, $0, Else
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)
```

Else:

```
0001 0010 0000 0000 0000 0000 0000 0011
1000 1110 0010 1000 0000 0000 0000 0100
0000 0001 0001 0000 0100 0000 0010 0000
1010 1110 0010 1000 0000 0000 0000 0000
```

3.4 Execution (Datapath)

- Given the binary
 - Assume two possible executions:
 1. $\$16 == \0 (*shorter*)
 2. $\$16 \neq \0 (*Longer*)
 - Convention:

Fetch: 

Memory: 

Decode: 

Reg Write: 

ALU: 

Other: 

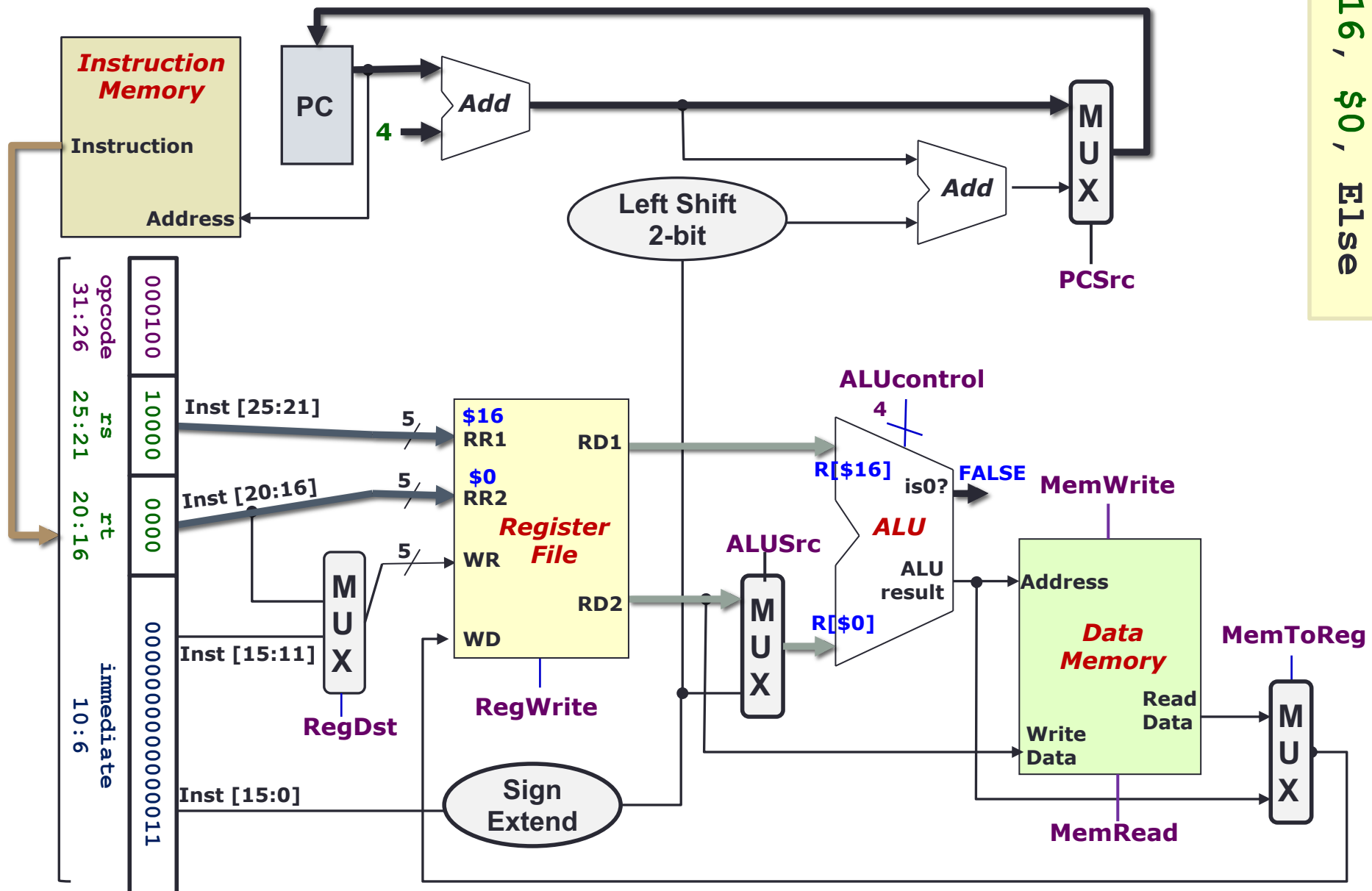
0001	0010	0000	0000	0000	0000	0000	0011
1000	1110	0010	1000	0000	0000	0000	0100
0000	0001	0001	0000	0100	0000	0010	0000
1010	1110	0010	1000	0000	0000	0000	0000

- ```
beq $16, $0, Else
```

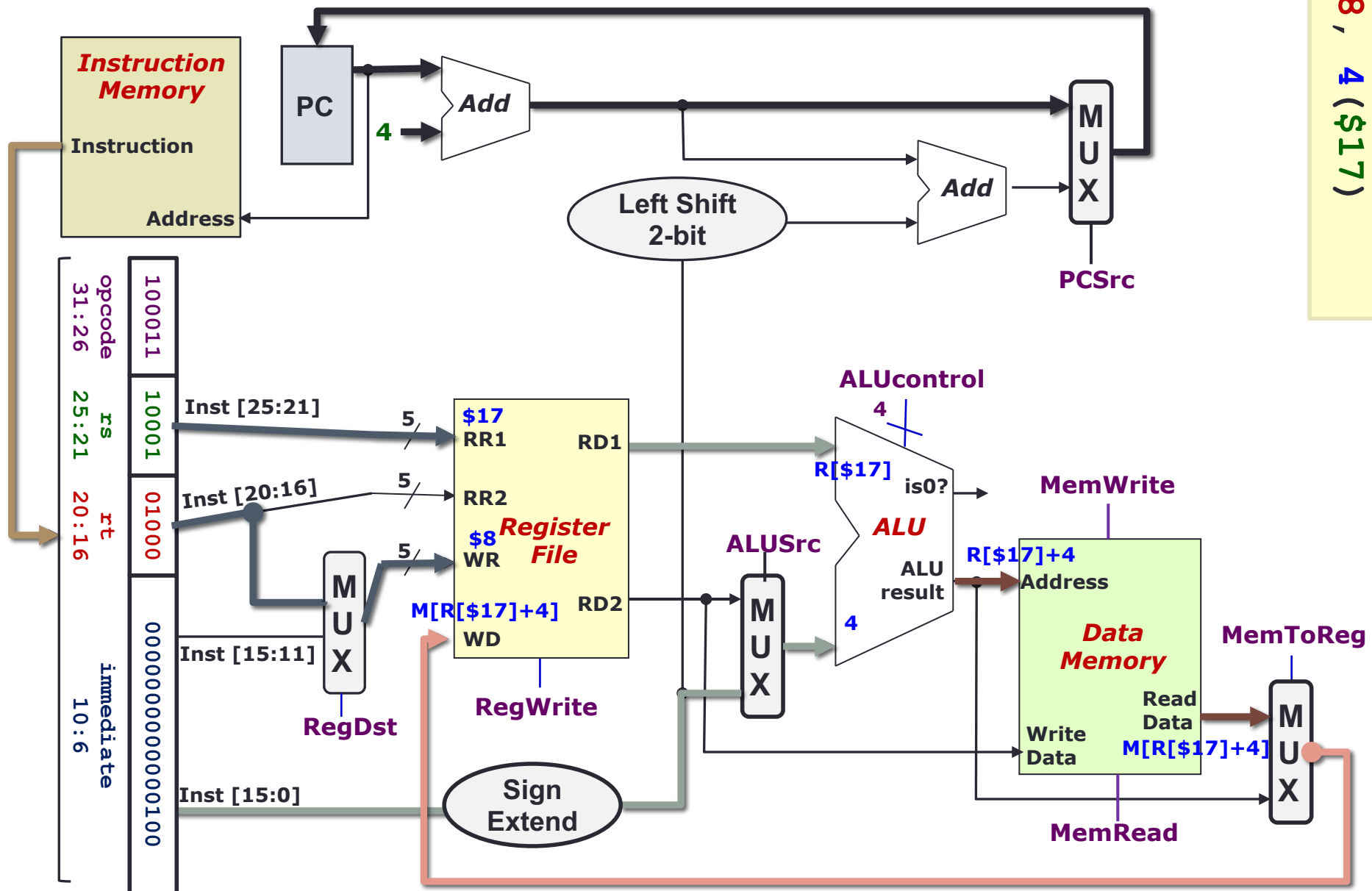




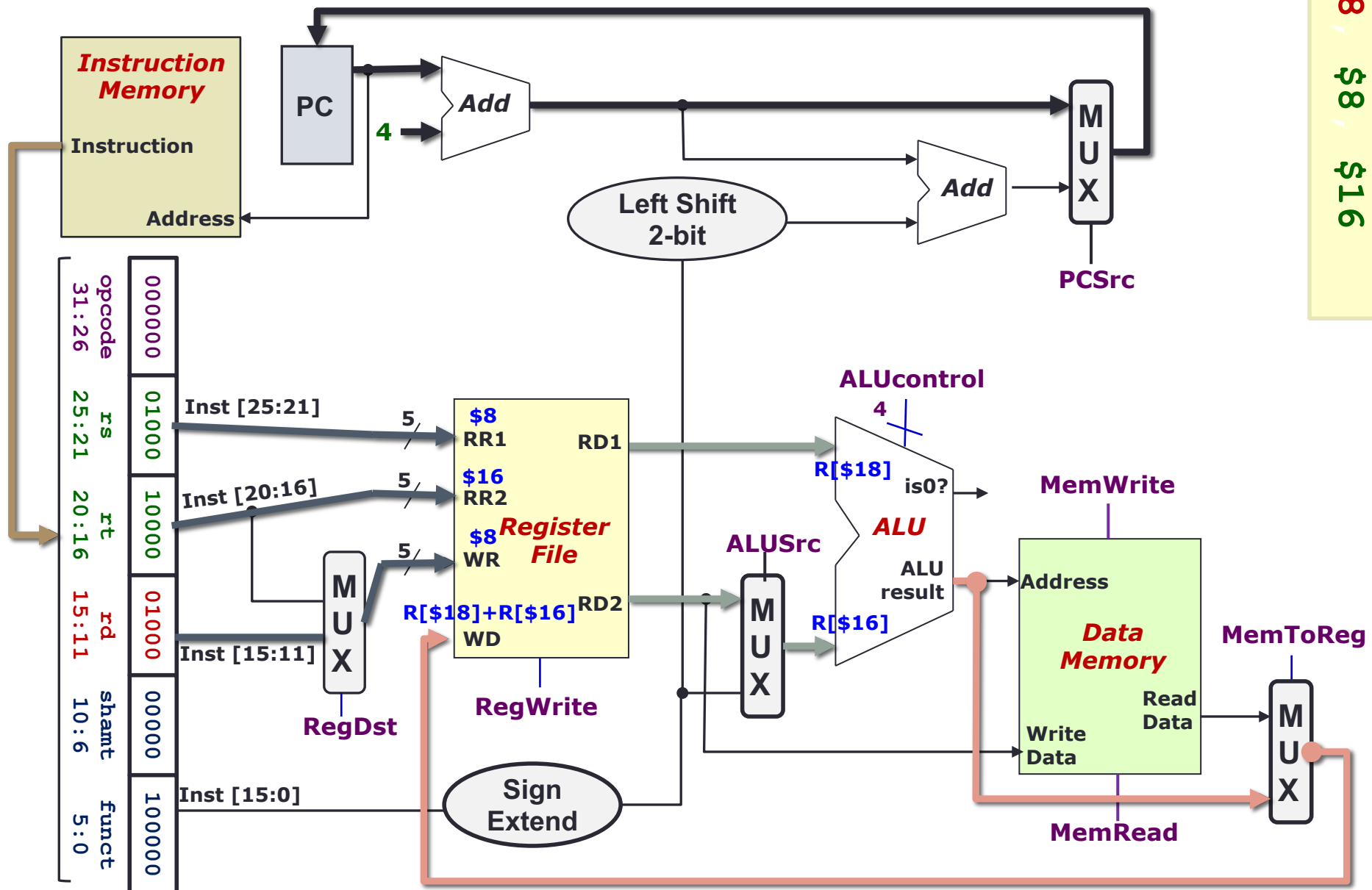
- Assume \$16 != \$0



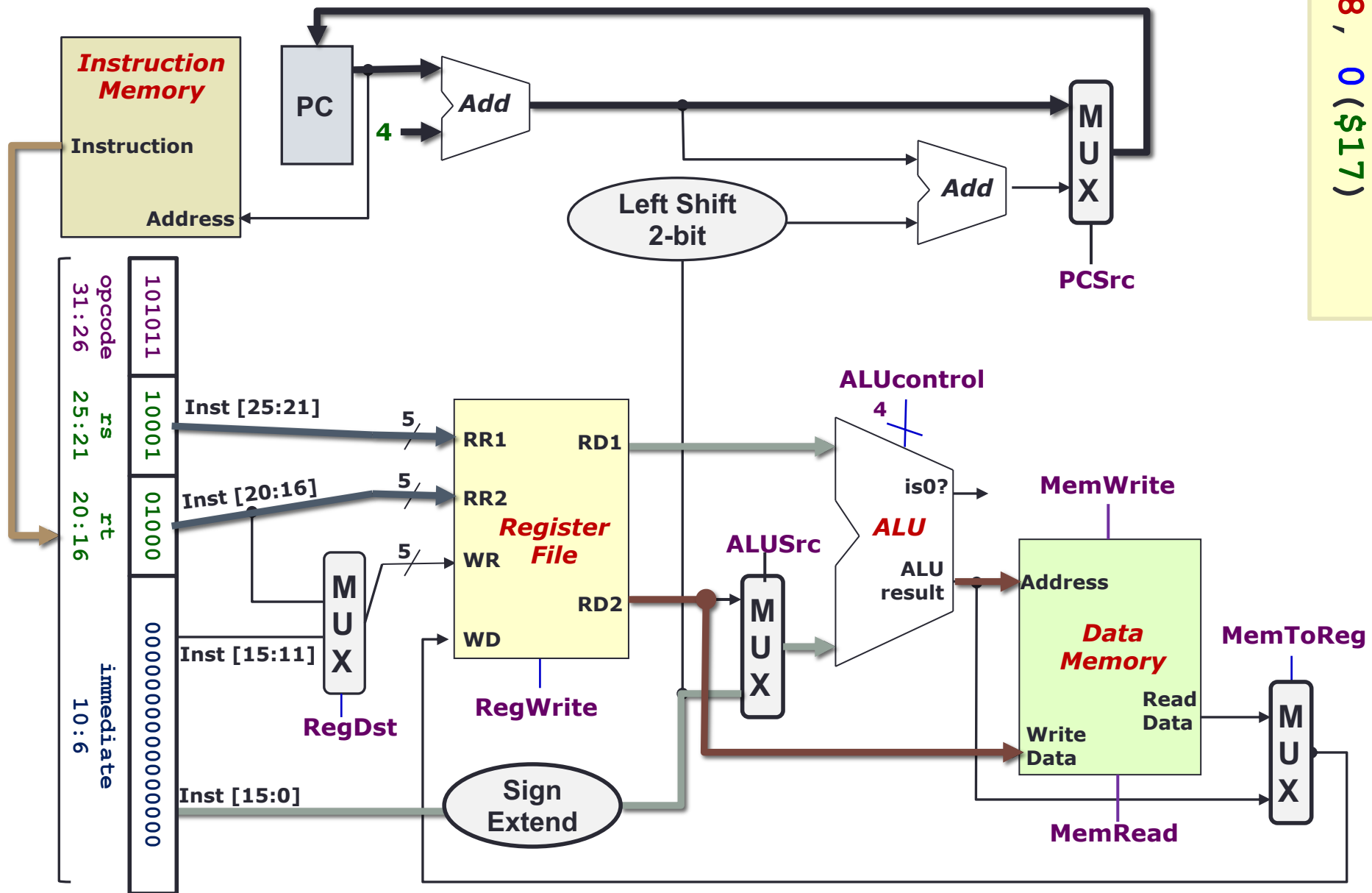
- Assume \$16 != \$0



- Assume \$16 != \$0



- Assume \$16 != \$0



End of File