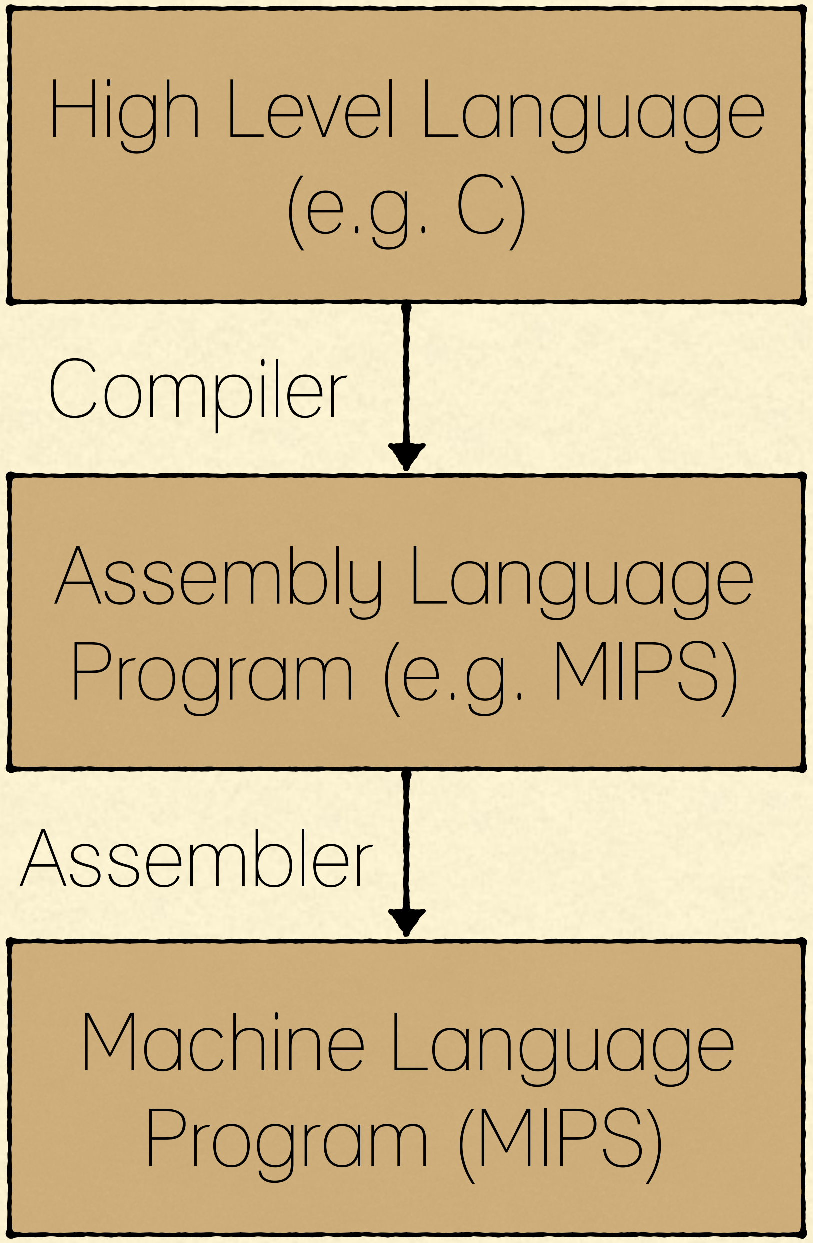
MIPS ASSEMBLY PROGRAMMING LANGUAGE PART III

Ayman Hajja, PhD

LEVELS OF REPRESENTATION



High Level Language (e.g. C)

Compiler

Assembly Language Program (e.g. MIPS)

Assembler

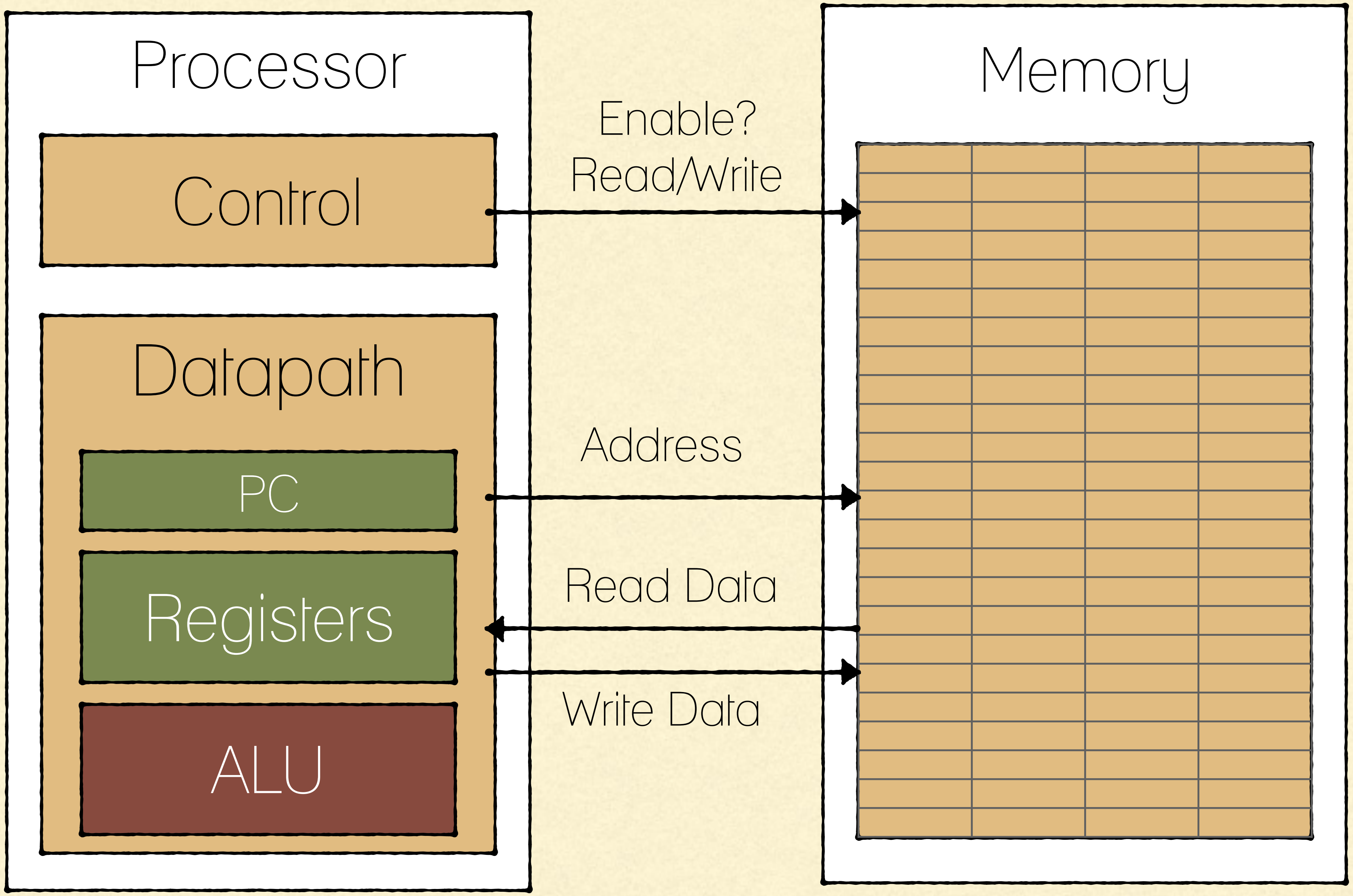
Machine Language Program (MIPS)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   $t0, 0($s2)
lw   $t1, 4($s2)
sw   $t1, 0($s2)
sw   $t0, 4($s2)
```

```
0000 1001 1001 0110 1010 1111 0101 1000
1111 1001 0000 1001 0000 1010 1111 0101
1001 1010 1111 0101 1000 1010 1111 0101
1001 1001 1010 1111 0101 1000 1010 1111
```

MEMORY ADDRESSES ARE IN BYTES



REGISTERS

- Registers are numbered from 0 to 31
- Each register can be referred to by a number or name
- Number references:
    - $0, $1, $2, …, $30, $31
- For now:
    - $16 to $23 will be referred to by $s0 to $s7 (variables)
    - $8 to $15 will be referred to by $t0 to $t7 (temp variables)
- In general, use names to make your code more readable

MIPS INSTRUCTIONS: ADDITION AND SUBTRACTION OF INTEGERS

- How to do the following C statement?

```
a = b + c + d - e;
```

- We break it into multiple instructions:

```
add $t0, $s1, $s2    # temp = b + c
add $t0, $t0, $s3    # temp = temp + d
sub $s0, $t0, $s4    # a = temp - e
```

a	$s0
b	$s1
c	$s2
d	$s3
e	$s4

IMMEDIATES

- Immediates are numerical constants that are embedded in the instruction itself
- Add Immediate:

```
addi $s0, $s1, -10 (in MIPS)

f = g - 10; (in C)
```

assuming $s0 and $s1 are associated with the variables f, g respectively

MIPS INSTRUCTIONS: LOAD WORD (LW)

- C code:

Offset (can't be a register)

  - int a[100];
  - g = h + a[3];
    - Assume the address of a is stored in $s3 (base register), and h is stored in $s2
- Using 'Load Word' (lw) in MIPS:
  - lw $t0, 12($s3)   # Temp reg $t0 gets a[3]
  - add $s1, $s2, $t0   # g = h + a[3]

MIPS INSTRUCTIONS: STORE WORD (SW)

- C code:

```
int a[100];
a[10] = h + a[3];
```

  - Assume the address of a is stored in $s3 (base register), and h is stored in $s2

- Steps:

```
lw $t0, 12($s3)         # Temp reg $t0 gets a[3]
add $t0, $s2, $t0       # temp = h + a[3]
sw $t0, 40($s3)         # a[10] = temp
```

MIPS INSTRUCTIONS: MORE OF LOADING/STORING

- In addition to word data transfers (lw, sw), MIPS has byte data transfers (and two bytes data transfers):
  - load byte; lb
  - store byte; sb
  - load half; lh
  - store half; sh

LOGIC SHIFTING

- Shift Left Logical: sll $s1, $s2, 2   # In C. s1 = s2 << 2;
  - Store in $s1 the value from $s2 shifted 2 bits to the left (they fall off end), inserting 0's on right; << in C
  - Before:

    0000 0000   0000 0000   0000 0000   0000 0010

  - After:

    0000 0000   0000 0000   0000 0000   0000 1000

COMPUTER DECISION MAKING

- Based on boolean expression (or condition), do something different
- In high level programming languages if/else statement
- In MIPS (branch on equal):

beq register1, register2, L1

If the value in register1 is equal to the value in register2, go to L1

COMPUTER DECISION MAKING

- There's also brach on not equal

bne register1, register2, L1

If the value in register1 is NOT equal to the value in register2, go to L1

TYPES OF BRANCHES

- Conditional Branch: change control flow depending on outcome of comparison
    - branch on equal (beq) or branch on not equal (bne)

- Unconditional Branch: always branch
    - MIPS instruction for this: jump
        - j Label

EXAMPLE IF STATEMENT

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

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

EXAMPLE IF STATEMENT

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

```
if (i == j)             bne    $s3, $s4, Exit
    f = g + h;          add    $s0, $s1, $s2
g = f + h;              Exit:  add   $s1, $s0, $s2
```

CONVERT C TO MIPS; QUESTION

```
int var_s0 = 10;
int var_s1 = 20;
// If both variables are equal; make both of them equal to 0
//
if (var_s0 == var_s1)
{
  var_s0 = 0;
  var_s1 = 0;
}
```

CONVERT C TO MIPS; SOLUTION

```c
int var_s0 = 10;

int var_s1 = 20;

if (var_s0 == var_s1)

{

   var_s0 = 0;

   var_s1 = 0;

}

// More code here
```

```
addi $s0, $0, 10

addi $s1, $0, 20

bne $s0, $s1, AFT

add $s0, $0, $0

add $s1, $0, $0

AFT:

# More code here
```

EXAMPLE IF STATEMENT

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

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

WHAT DO YOU THINK OF THIS SOLUTION?

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

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

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

WHAT DO YOU THINK OF THIS SOLUTION?

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

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

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

WRONG ANSWER

EXAMPLE IF STATEMENT

f	g	h	i	j
$s0	$s1	$s2	$s3	$s4

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

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

MIPS LOGICAL INSTRUCTIONS

- logical bitwise operators (two registers):
  - Bitwise AND — and $rd, $rs, $rt
  - Bitwise OR — or $rd, $rs, $rt
  - Bitwise NOR — nor $rd, $rs, $rt
  - Bitwise XOR — xor $rd, $rs, $rt
- logical bitwise operators (one register and one immediate):
  - andi $rt, $rs, immed
  - ori $rt, $rs, immed
  - xori $rt, $rs, immed

MIPS LOGICAL INSTRUCTIONS

- logical bitwise operators (two registers):
  - Bitwise AND — and $rd, $rs, $rt
  - Bitwise OR — or $rd, $rs, $rt
  - Bitwise NOR — nor $rd, $rs, $rt
  - Bitwise XOR — xor $rd, $rs, $rt
- logical bitwise operators (one register and one immediate):
  - andi $rt, $rs, immed
  - ori $rt, $rs, immed
  - xori $rt, $rs, immed

How can we get bitwise NOT?

One way is to NOR value with itself

MIPS LOGICAL INSTRUCTIONS

- logical bitwise operators (two registers):
  - Bitwise AND — and $rd, $rs, $rt
  - Bitwise OR — or $rd, $rs, $rt
  - Bitwise NOR — nor $rd, $rs, $rt
  - Bitwise XOR — xor $rd, $rs, $rt
- logical bitwise operators (one register and one immediate):
  - andi $rt, $rs, immed
  - ori $rt, $rs, immed
  - xori $rt, $rs, immed

How can we get bitwise NOT?

Another way would be to NOR with 0

MIPS LOGICAL INSTRUCTIONS

- logical bitwise operators (two registers):
    - Bitwise AND — and $rd, $rs, $rt
    - Bitwise OR — or $rd, $rs, $rt
    - Bitwise NOR — nor $rd, $rs, $rt
    - Bitwise XOR — xor $rd, $rs, $rt
- logical bitwise operators (one register and one immediate):
    - andi $rt, $rs, immed
    - ori $rt, $rs, immed
    - xori $rt, $rs, immed

How can we get bitwise NOT?

Also, we can xor (or xori) the value of the register with -1