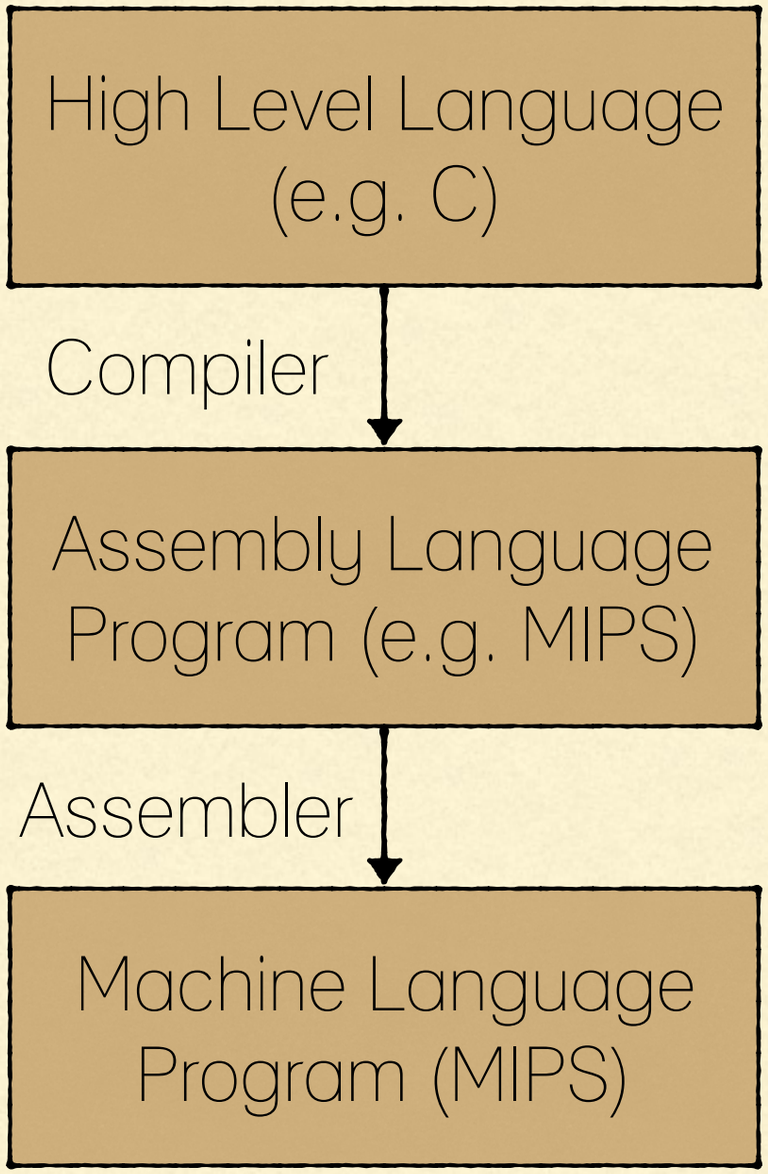
MIPS ASSEMBLY PROGRAMMING LANGUAGE PART II

Ayman Hajja, PhD

LEVELS OF REPRESENTATION

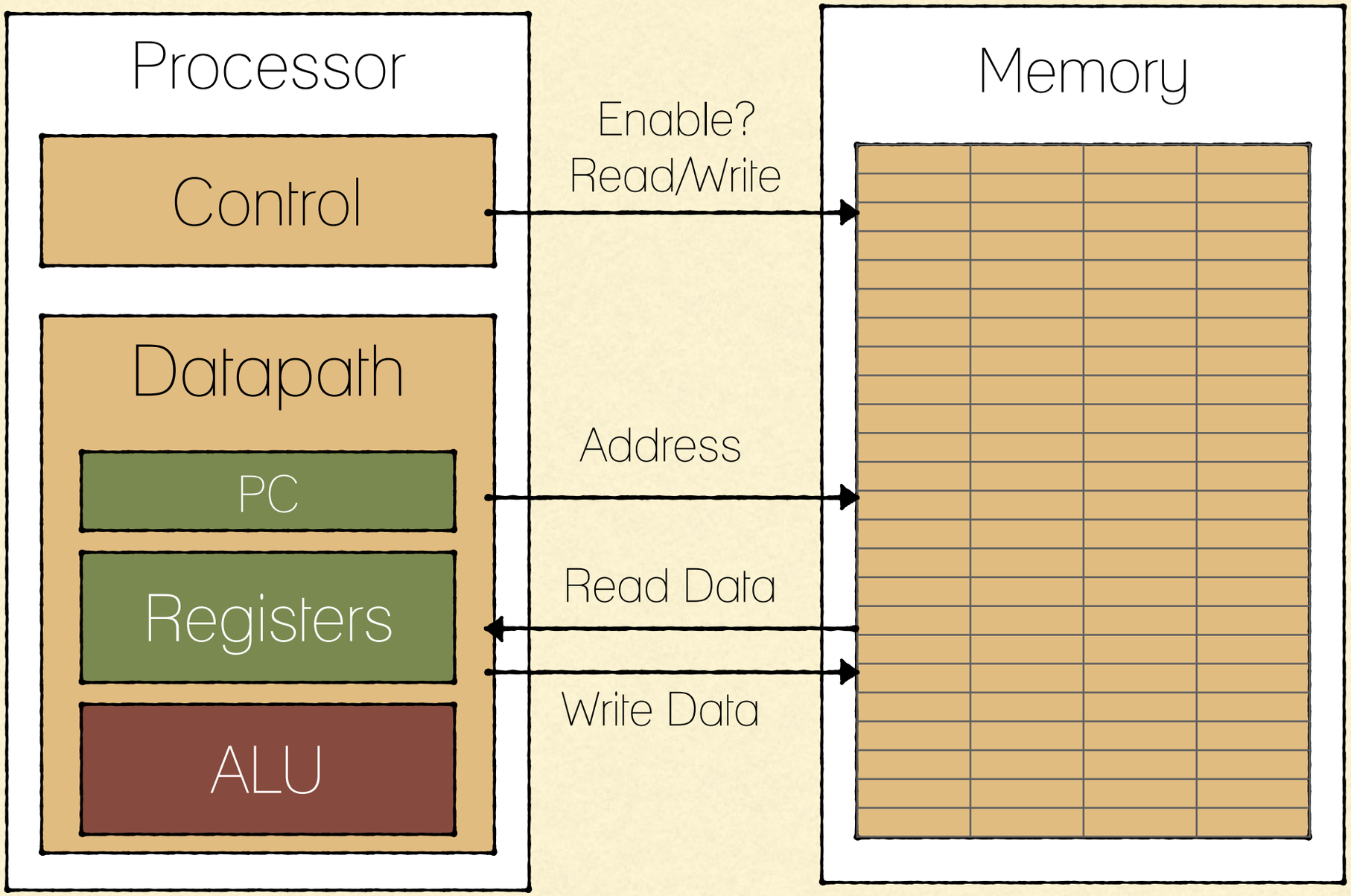



```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

```
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: ADD (REGISTER INSTRUCTION)

- Addition in Assembly:
  - Example 1: add $s0, $s1, $s2 (in MIPS)
  - Equivalent to a = b + c; (in C), assuming that:
    - $s1 contains the value of b
    - $s2 contains the value of c
    - and $s0 will be used to store the result (equivalent to 'a')

MIPS INSTRUCTIONS: ADD (REGISTER INSTRUCTION)

- Addition in Assembly:
  - Example 1: add $s0, $s1, $s2 (in MIPS)
  - Equivalent to a = b + c; (in C), assuming that:
    - $s1 contains the value of b
    - $s2 contains the value of c
    - and $s0 will be used to store the result (equivalent to 'a')
  - Example 2: add $s0, $s1, $zero (in MIPS)
  - Equivalent to f = g; (in C), assuming that $s0 corresponds to f, and $s1 corresponds to g

MIPS INSTRUCTIONS: SUB (REGISTER INSTRUCTION)

- Subtraction in Assembly:
  - Example: sub $s3, $s4, $s5 (in MIPS)
  - Equivalent to: d = e - f; (in C), assuming that:
    - d corresponds to $s3
    - e corresponds to $s4
    - f corresponds to $s5

MIPS INSTRUCTIONS: ADDITION AND SUBTRACTION OF INTEGERS

- How to do the following C statement?

  a = b + c + d - e;

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

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
```

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

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
```

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

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:
  - int a[100];
  - g = h + a[3];
    - Assume the address of a is stored in $s3 (base register), and h is stored in $s2

MIPS INSTRUCTIONS: LOAD WORD (LW)

- C code:
  - 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]

Offset (can't be a register)

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:

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]
```

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]
```

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

QUESTION

- How would the following code translates to MIPS:
  - *x = *y + 1
- Assuming x & y pointers are stored in $s0 & $s1 respectively)

A) addi $s0, $s1, 1

B) lw $s0, 1($s1)
   sw $s1, 0($s0)

C) lw $t0, 0($s1)
   addi $t0, $t0, 1
   sw $t0, 0($s0)

D) sw $t0, 0($s1)
   addi $t0, $t0, 1
   lw $t0, 0($s0)

QUESTION

- How would the following code translates to MIPS:
  - *x = *y + 1
- Assuming x & y pointers are stored in $s0 & $s1 respectively)

A) addi $s0, $s1, 1

B) lw $s0, 1($s1)
sw $s1, 0($s0)

C) lw $t0, 0($s1)
addi $t0, $t0, 1
sw $t0, 0($s0)

D) sw $t0, 0($s1)
addi $t0, $t0, 1
lw $t0, 0($s0)

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

Question:
How can we get bitwise NOT?

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 do we can we get bitwise NOT?
Set a register to -1 (add immediate)
xor the value with that register