INSTRUCTION SET ARCHITECTURE

Mahdi Nazm Bojnordi

Assistant Professor

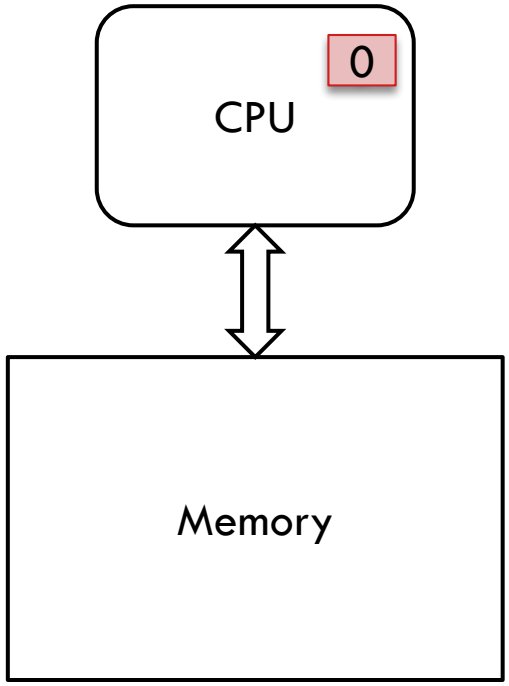
School of Computing

University of Utah

Constant Values

- Constant values are defined/used in code
  - Known to the programmer
  - Zero is commonly used

```
 8 int main() {
 9     int i, j;
10     for(j = 0; j < 10; j ++) {
11         for(i = 0; i < mem_size >> 2; i += 16) {
12             p[i] = 55;
13         }
14         for(i = 0; i < mem_size >> 2; i += 16) {
15             q[i] = 56;
16         }
17     }
18     return 0;
19 }
```



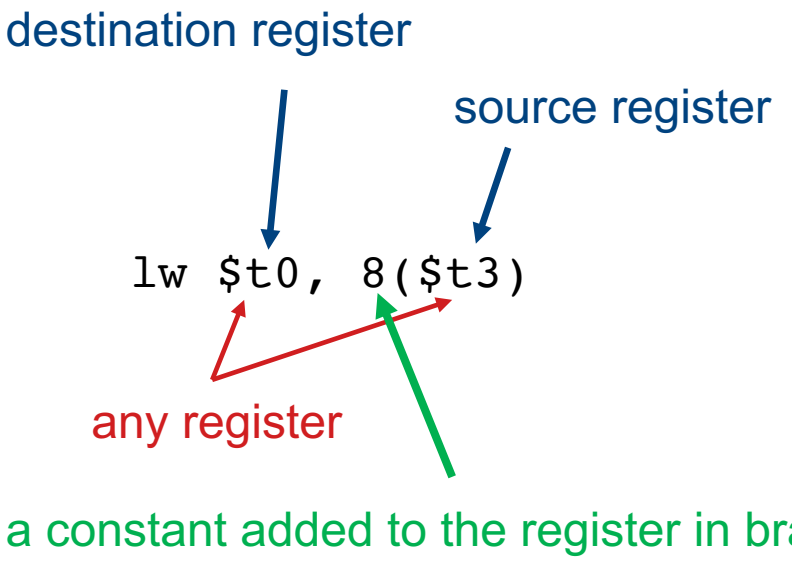
How to handle constants in the ISA?

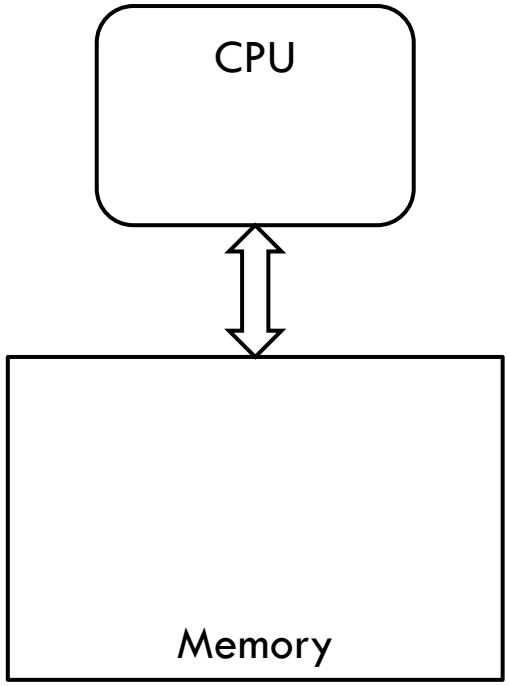
Immediate Operand

- An instruction may require a constant as input
- An immediate instruction uses a constant number as one of the inputs (instead of a register operand)
- Putting a constant in a register requires addition to register $zero (a special register that always has zero in it) -- since every instruction requires at least one operand to be a register
- For example, putting the constant 1000 into a register:
  - addi $s0, $zero, 1000

Memory Instruction Format

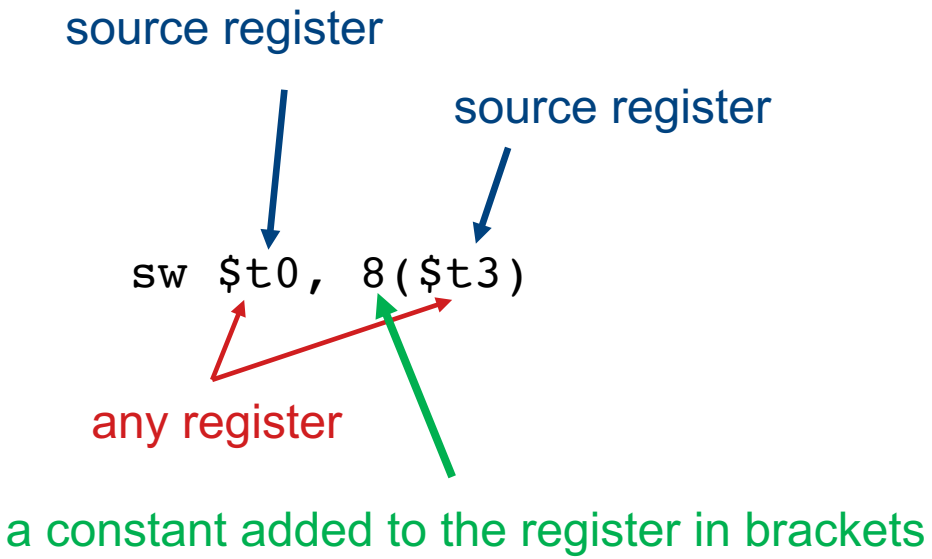
- The format of a load instruction:

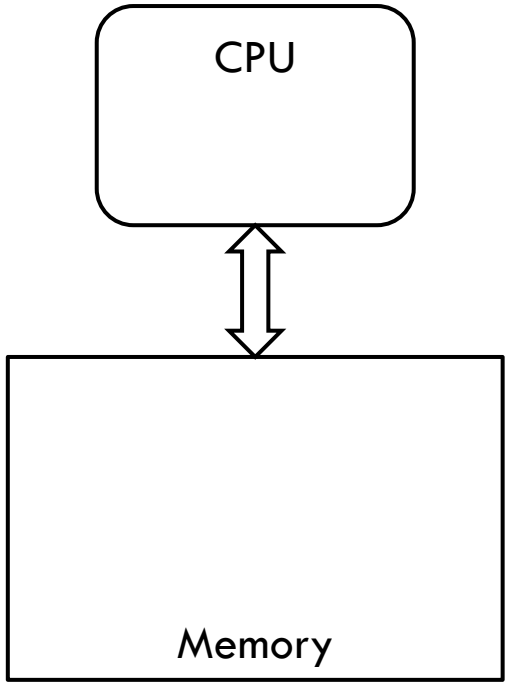




Memory Instruction Format

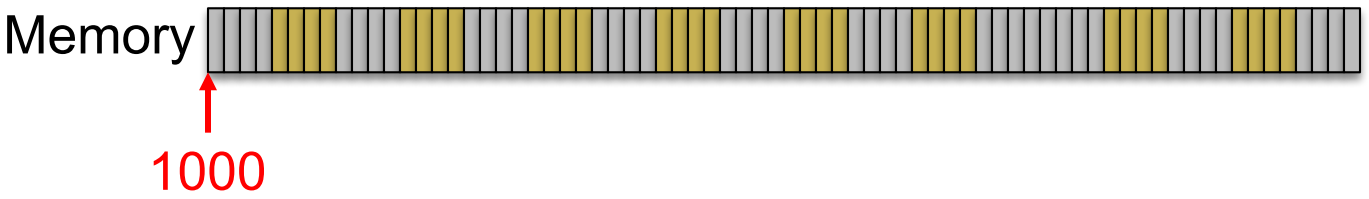
- The format of a load instruction:





Example MIPS Translation

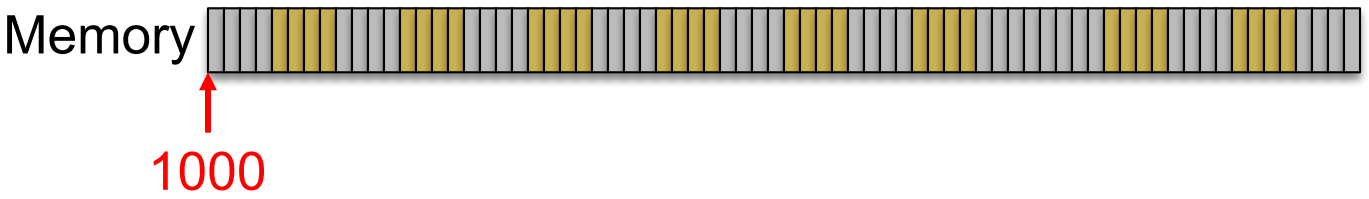
- int a, b, c, d[10]



- Task: bring a, b, c, d[0], and d[1] to $s1-$s5

Example MIPS Translation

- int a, b, c, d[10]

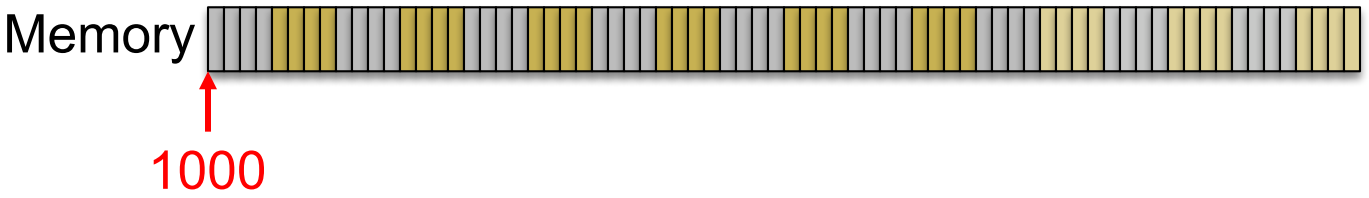


- Task: bring a, b, c, d[0], and d[1] to $s1-$s5

```
addi   $t0, $zero, 1000   # put base address 1000 in $t0;
                          # $zero is a register that always equals zero
```

Example MIPS Translation

- int a, b, c, d[10]



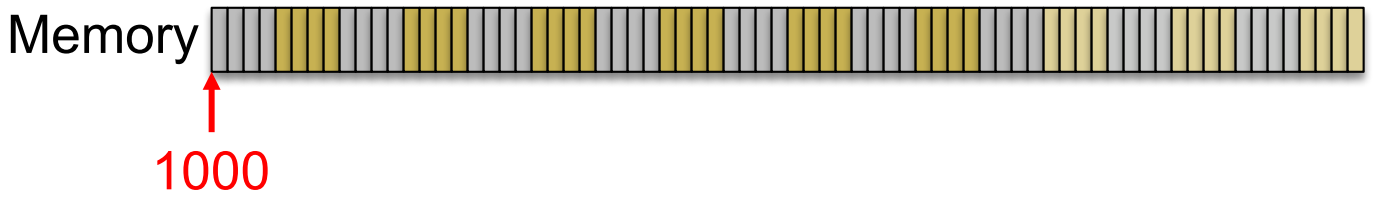
- Task: bring a, b, c, d[0], and d[1] to $s1-$s5

```
addi   $t0, $zero, 1000   # put base address 1000 in $t0;
                          # $zero is a register that always equals zero

lw   $s1, 0($t0)          # brings value of a into register $s1
lw   $s2, 4($t0)          # brings value of b into register $s2
lw   $s3, 8($t0)          # brings value of c into register $s3
lw   $s4, 12($t0)         # brings value of d[0] into register $s4
lw   $s5, 16($t0)         # brings value of d[1] into register $s5
```

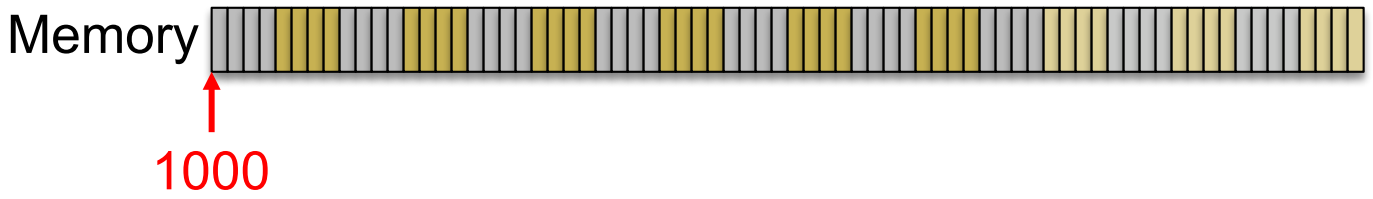
Example MIPS Translation

- Convert the following C code to assembly
  - d[3] = d[2] + a;



Example MIPS Translation

- Convert the following C code to assembly
  - d[3] = d[2] + a;



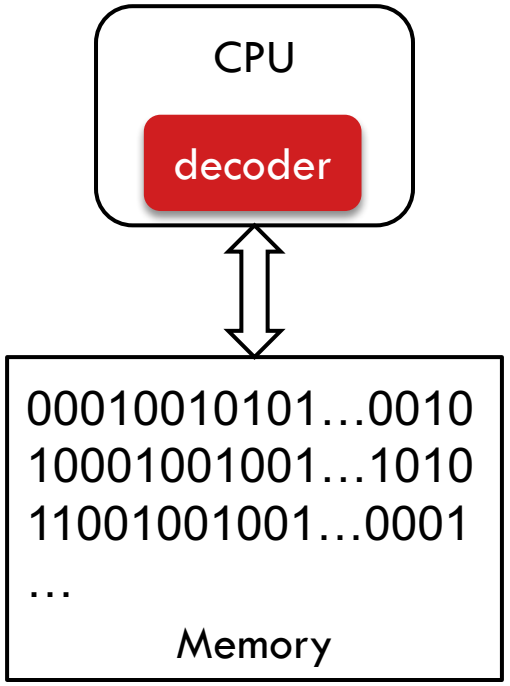

```
addi   $t0, $zero, 1000   # put base address 1000 in $t0;
                          # $zero is a register that always equals zero

lw     $s0, 0($t0)        #  a is brought into $s0
lw     $s1, 20($t0)       #  d[2] is brought into $s1
add    $t1, $s0, $s1      #  the sum is in $t1
sw     $t1, 24($t0)       #  $t1 is stored into d[3]
```

Instruction Formats

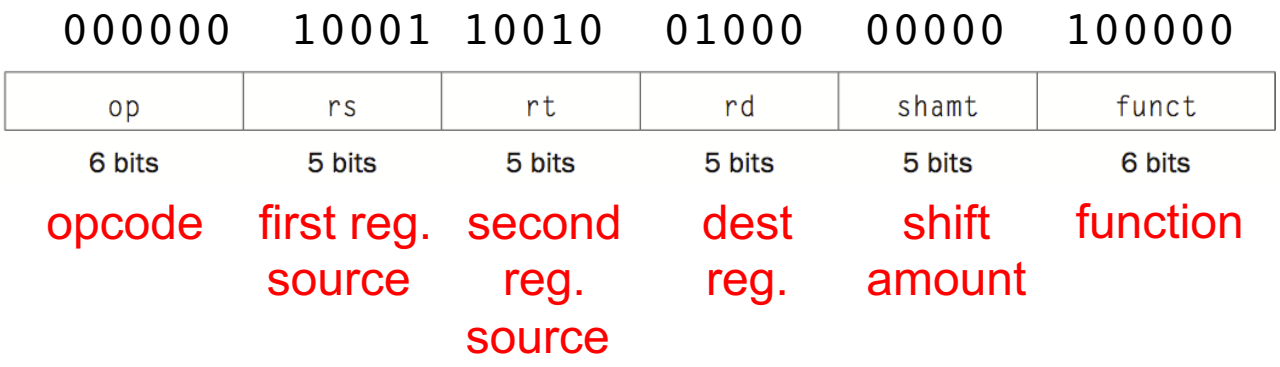
- Instructions are represented as 32-bit numbers
  - Each instruction word has multiple fields
- MIPS Instruction Types
  - R-type
    - add $t0, $s1, $s2

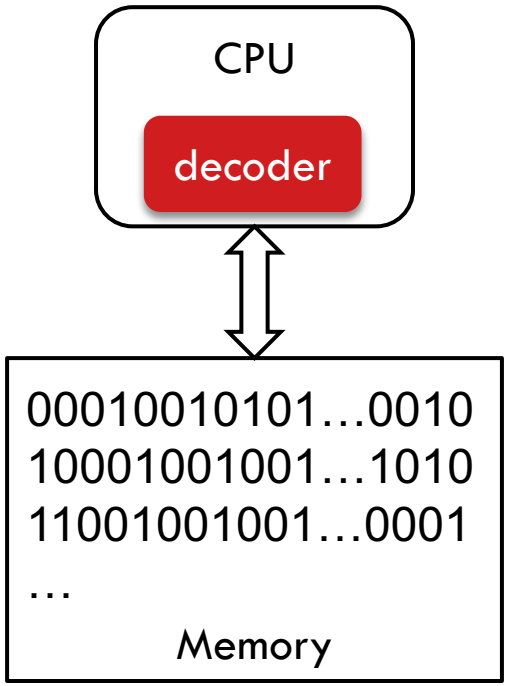
```
000000  10001 10010  01000  00000  100000
```



Instruction Formats

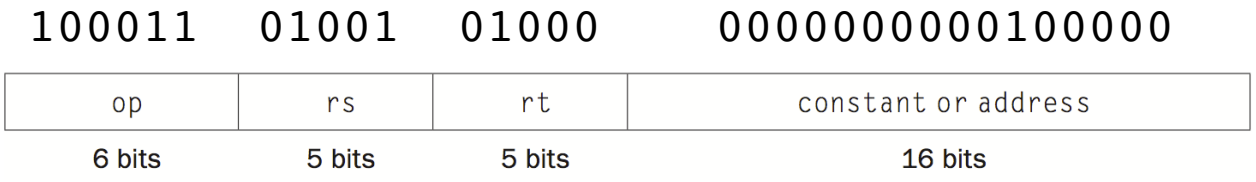
- Instructions are represented as 32-bit numbers
  - Each instruction word has multiple fields
- MIPS Instruction Types
  - R-type
    - add $t0, $s1, $s2

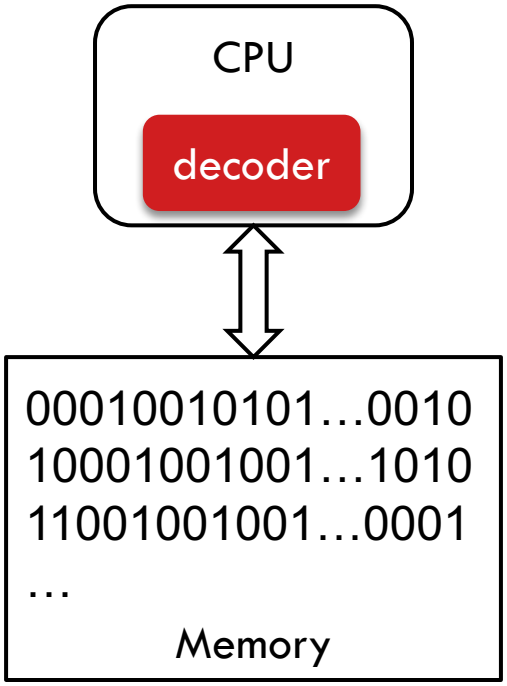




Instruction Formats

- Instructions are represented as 32-bit numbers
  - Each instruction word has multiple fields
- MIPS Instruction Types
  - R-type
    - add $t0, $s1, $s2
  - I-type
    - lw $t0, 32($t1)





Logical Operations

- Bitwise logical operations

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

- Shift
  - sll $t2, $s0, 4
  - srl $t2, $s0, 4

Logical Operations

- Bitwise logical operations

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

- Shift
- AND
  - and $t0, $t1, $t2

Logical Operations

- Bitwise logical operations

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

- Shift
- AND
- OR
  - or $t0, $t1, $t2

Logical Operations

- Bitwise logical operations

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

- Shift
- AND
- OR
- NOT
  - nor $t0, $t1, $t2