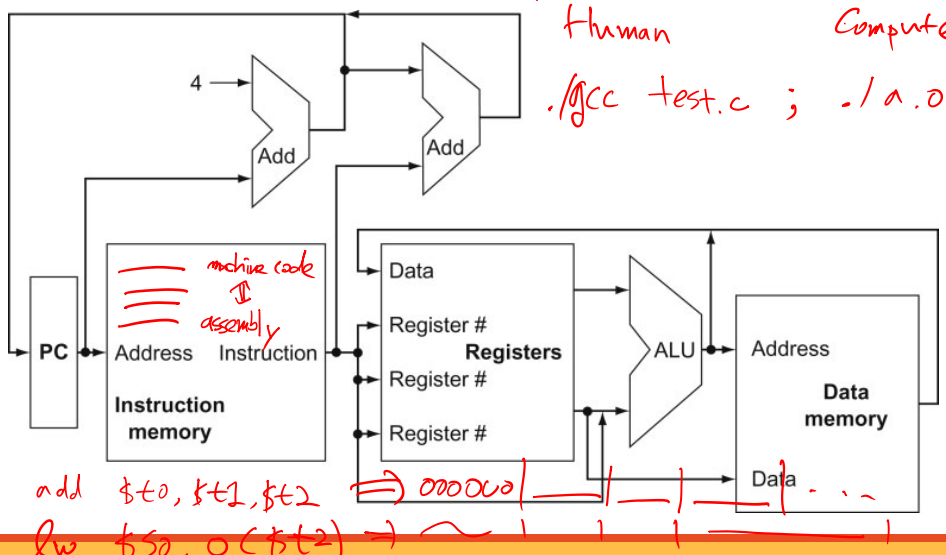
Big Picture



Representing Instructions

- Instructions are encoded in binary
  - Called machine code
- MIPS instructions
  - Encoded as 32-bit instruction words
  - Small number of formats encoding operation code (opcode), register numbers, …
  - Regularity!
- Register numbers
  - $t0 – $t7 are reg's
  - $t8 – $t9 are reg's
  - $s0 – $s7 are reg's

MIPS R-format Instructions

- Instruction fields
  - op: operation code
  - rs: 1st source register
  - rt: 2nd source register
  - rd: destination
  - shamt: shift amount
  - funct: function code

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

R-format example

- `add $t0, $s1, $s2` ← Human

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
0	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

← computer

32 bits

$?_2 = 02324020_{16}$

Hexadecimal

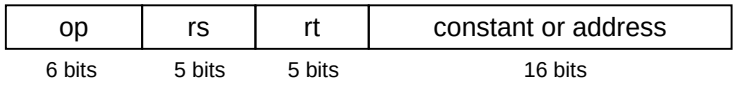
- Base 16
  - Compact representation of bit strings
  - 4 bits per hex digit
- Example: eca8 6420
  - 1110 1100 1010 1000

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

9/15

MIPS I-format Instructions

- Immediate arithmetic and load/store instructions
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15} - 1$
  - Address: offset added to base address in rs
- Design Principle 4: Good design demands good compromises
  - Different formats complicate decoding, but allow 32-bit instructions uniformly
  - Keep formats as similar as possible



I-format example

- `lw $t0, 16($s1)`

int[] A;
&(A[0]) → $s1
| 0 | 1 | 2 | 3 | 4 |
$s1 +4 +8 +12 +16

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits
35	17	8	16
10011	10001	01000	0000 0000 0001 0000

How about larger constants?

- Can we load a 32-bit constant by using a single instruction?
  - ~~Yes~~ .
- lui: load upper immediate

```
lui $t0, 1010101010101010
```

1010 ~ 1010	0000 ~ 0000
16 bits	16 bits

- ori: bitwise or immediate

```
ori $t0, 0101010101010101
```

1010 ~ 1010	0101 ~ 0101
16 bits	16 bits

addi $t0, 16

or $t0, $t1, $t2
$t0, $t0, 0101~0101

add $t1 $t0 $t0

1010 ~ 1010 0101 ~ 0101
+ 1010 ~ 1010 0101 ~ 0101 ?

Logical Operations

- Instructions for bitwise manipulation

Operation	C	MIPS
Shift left	<<	sll
Shift right	>>	srl
Bitwise AND	&	and , andi
Bitwise OR	|	or , ori
Bitwise NOT	~	nor

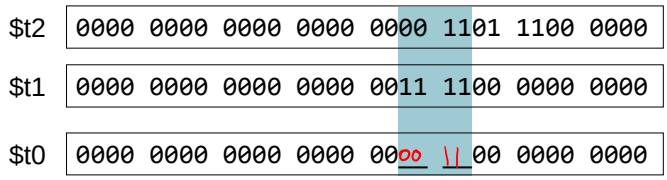
Shift Operations

- shamt: how many positions to shift
- Shift left logical
  - Shift left and fill with 0 bits
  - *sll* by *i* bits multiplies by 2^i
- Shift right logical
  - Shift right and fill with 0 bits
  - *srl* by *i* bits divides by 2^i (unsigned only)
- Shift right arithmetic(sra) for signed.

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

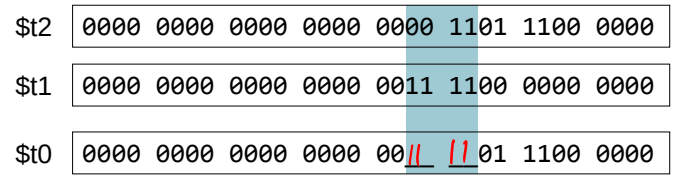
AND Operations

- Useful to mask bits in a word
  - Select some bits, clear others to 0
- `and $t0, $t1, $t2`



OR Operations

- Useful to include bits in a word
  - Set some bits to 1, leave others unchanged
- `or $t0, $t1, $t2`



NOT Operations

- Useful to invert bits in a word
  - Change 0 to 1, and 1 to 0
- MIPS has NOR 3-operand instruction
- a NOR b == NOT (a OR b)
- nor $t0, $t1, $zero

Register 0: always read as zero

$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	1111 1111 1111 1111 1100 0011 1111 1111

Conditional Operations

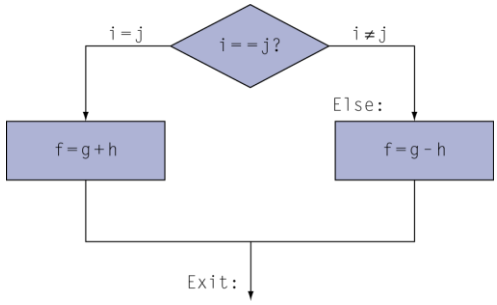
- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially
- beq rs, rt, L1
  - if (rs == rt) branch to instruction labeled L1;
- bne rs, rt, L1
  - if (rs != rt) branch to instruction labeled L1;
- j L1
  - unconditional jump to instruction labeled L1

if (rs == rt) {
}
else {
}

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: …
```



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: …
```