CHAPTER 2-3

Instruction Language of the Computer

By Pattama Longani
Collage of arts, media and Technology

ASCII

- Most computers today offer 8-bit bytes to represent characters, with American Standard Code for Information Interchange (ASCII). ASCII is represented decimal mapping as the table below.

ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

EX: Dog

D = 68_{10} = $0100\ 0100_2$ = 44_{16}

o = 111_{10} = $0110\ 1111_2$ = $6F_{16}$

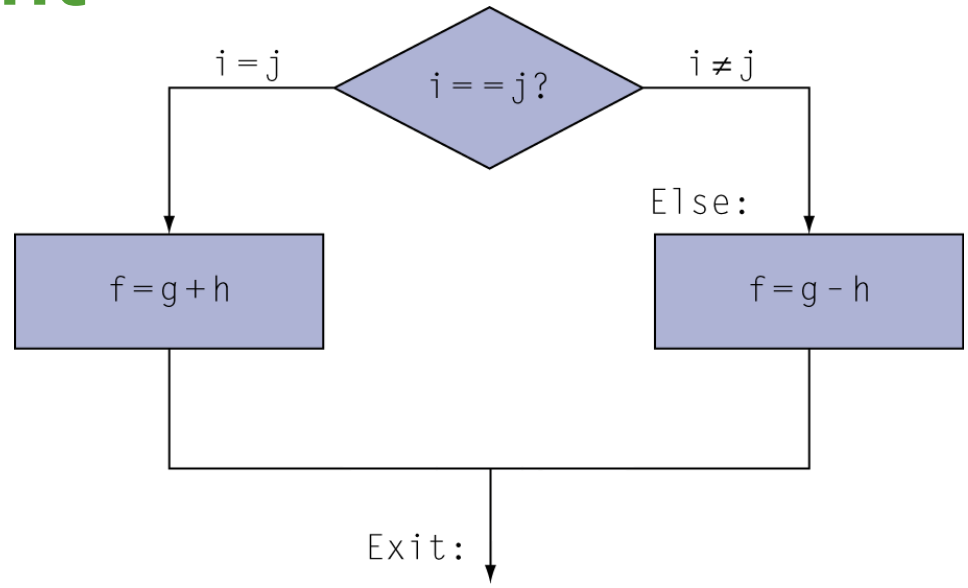
g = 103_{10} = $0110\ 0111_2$ = 67_{16}

CONDITIONAL OPERATIONS

- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially
- `beq rs, rt, L1` **(branch if equal)**
  - if (rs == rt) branch to instruction labeled `L1`;
- `bne rs, rt, L1` **(branch if not equal)**
  - if (rs != rt) branch to instruction labeled `L1`;
- `j L1`
  - unconditional jump to instruction labeled `L1`

•C code: If statement

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



Suppose f, g are in $s0, $s1, compiled MIPS code:

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

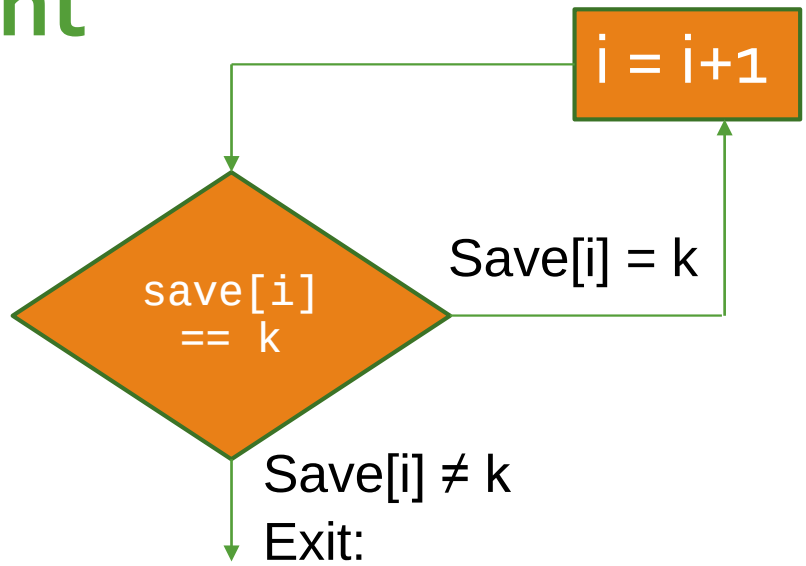
Assembler calculates addresses

•C code: Loop Statement

```
while (save[i] == k){

     i += 1;

}
```



Suppose 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: …
```

slt = *set on less*

- Set result to 1 if a condition is true
  - Otherwise, set to 0
- `slt rd, rs, rt`
  - if (rs < rt) rd = 1; else rd = 0;
- `slti rt, rs, constant`
  - if (rs < constant) rt = 1; else rt = 0;
- Use in combination with `beq`, `bne`

```
slt $t0, $s1, $s2  # if ($s1 < $s2)
bne $t0, $zero, L  # branch to L
```

Signed vs. Unsigned

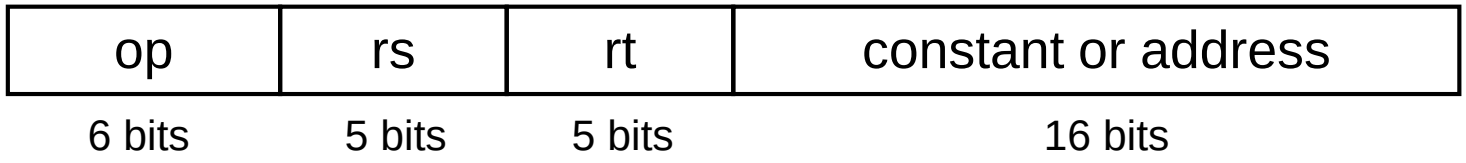
- `slt, slti` are Signed comparison
- `sltu, sltui` are Unsigned comparison

EXAMPLE

- $s0 = 1111 1111 1111 1111 1111 1111 1111 1111
- $s1 = 0000 0000 0000 0000 0000 0000 0000 0001
- `slt $t0, $s0, $s1 # signed`
  - −1 < +1 ⇒ $t0 = 1
- `sltu $t0, $s0, $s1 # unsigned`
  - +4,294,967,295 > +1 ⇒ $t0 = 0

Branch Addressing (I-format)

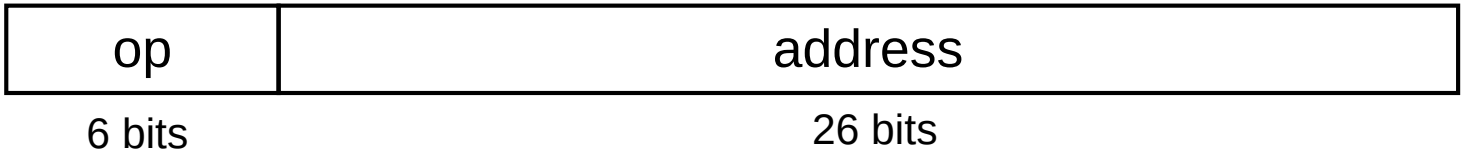
- Branch instructions specify
  - Opcode, two registers, target address
- Most branch targets are near branch
  - Forward or backward



- PC-relative addressing
  - Target address = PC + offset × 4
  - PC already incremented by 4 by this time

J-Type instruction

- Jump (`j` and `jal`) targets could be anywhere in text segment
  - Encode full address in instruction



- (Pseudo)Direct jump addressing
  - Target address = $PC_{31\ldots28}$: (address × 4)

Target Addressing Example

- Loop code from earlier example
  - Assume Loop at location 80000

```
Loop: sll  $t1, $s3, 2      80000
      add  $t1, $t1, $s6    80004
      lw   $t0, 0($t1)      80008
      bne  $t0, $s5, Exit   80012
      addi $s3, $s3, 1      80016
      j    Loop             80020
Exit: …                     80024
```

20000*4

80016+(2*4)

0	0	19	9	2	0
0	9	22	9	0	32
35	9	8	0		
5	8	21	2		
8	19	19	1		
2	20000				