NUMBER REPRESENTATION

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

THE UNIVERSITY OF UTAH

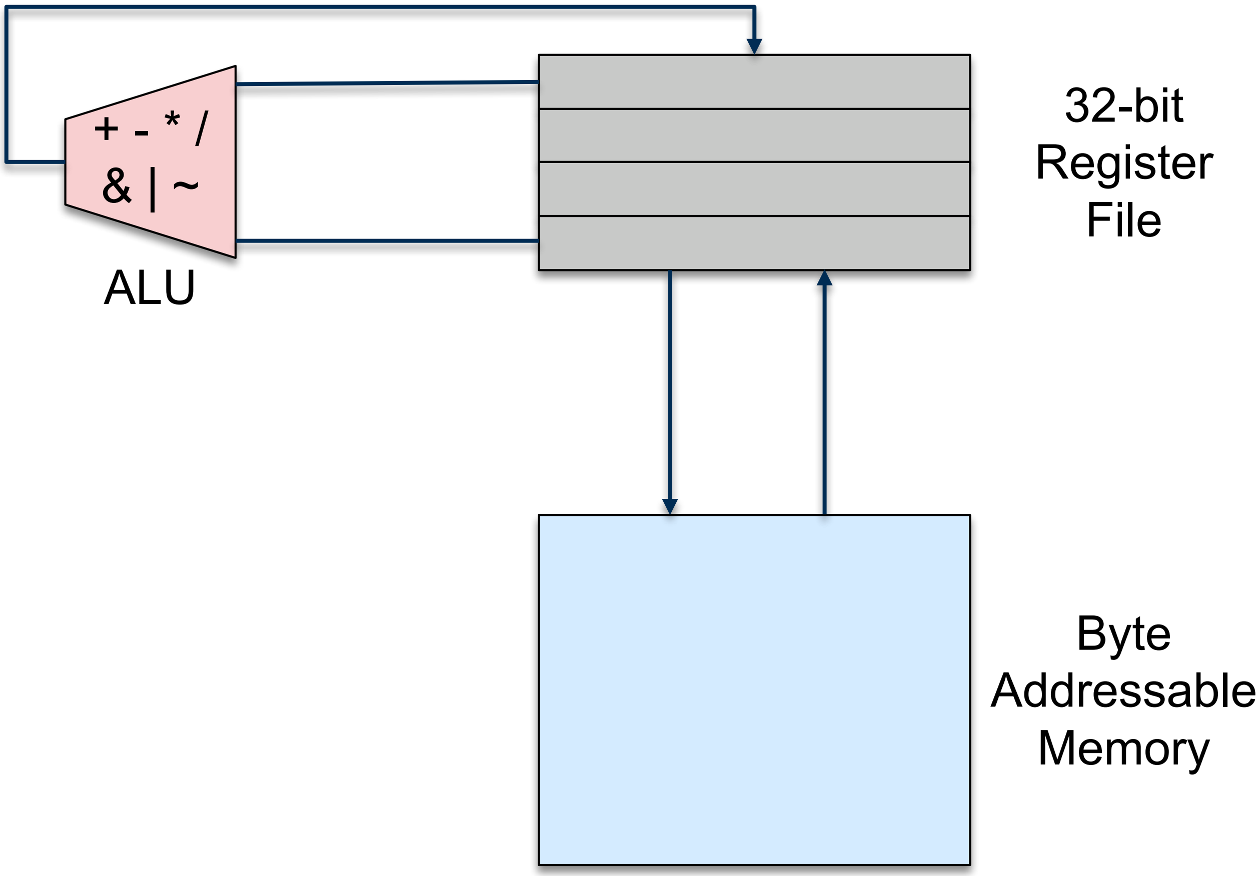
CS/ECE 3810: Computer Organization

Overview

- This lecture
  - Dealing with characters
  - Large constants
  - Binary representation
  - Negative numbers
  - Signed vs. unsigned

The Big Picture So Far

- Functional units, register file, and memory



Dealing with Characters

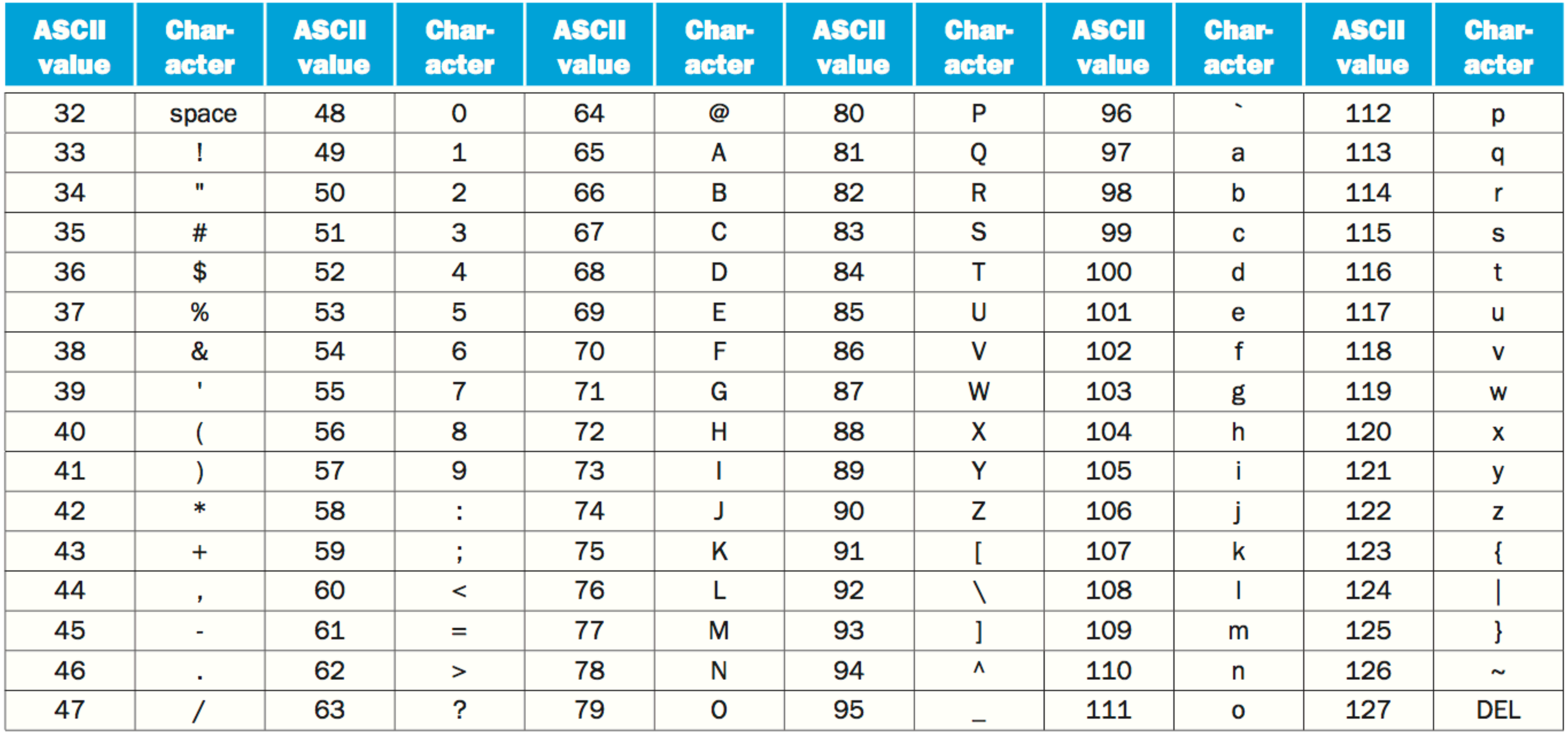
- Instructions are also provided to deal with byte-sized and half-word quantities: lb (load-byte), sb (store-byte), lh (load-half-word), sh (store-half-word)

- These data types are most useful when dealing with characters, pixel values, etc.
  - e.g., `printf(“Hello World!”);`

- C employs ASCII formats to represent characters – each character is represented with 8 bits and a string ends in the null character
  - e.g., null is 0, A is 65, a is 97

Dealing with Characters

- ASCII: American Standard Code for Information Interchange



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

Example: String Copy

- Convert to Assembly

```
void strcpy (char x[], char y[])
{
    int i=0;
    while  ((x[i] = y[i]) != `\0’)
        i += 1;
}
```

Example: String Copy

- Convert to Assembly

```
                  $a0      $a1
void strcpy (char x[], char y[])
{    $s0
    int i=0;
    while  ((x[i] = y[i]) != `\0’)
        i += 1;
}
```

Callee saves $s0-$s7

Example: String Copy

- Convert to Assembly

```
                 $a0       $a1
void strcpy (char x[], char y[])
{     $s0
    int i=0;
    while  ((x[i] = y[i]) != `\0’)
        i += 1;
}
```

```
strcpy:
        addi $sp, $sp, -4
        sw   $s0, 0($sp)
        add  $s0, $zero, $zero
```

Example: String Copy

□ Convert to Assembly

```
                    $a0      $a1
void strcpy (char x[], char y[])
{     $s0
   int i=0;
   while  ((x[i] = y[i]) != `\0’)
      i += 1;
}
```

```
strcpy:
        addi $sp, $sp, -4
        sw   $s0, 0($sp)
        add  $s0, $zero, $zero
while:  add  $t1, $s0, $a1
        lb   $t2, 0($t1)
        add  $t3, $s0, $a0
        sb   $t2, 0($t3)
```

Example: String Copy

□ Convert to Assembly

```
                        $a0     $a1
void strcpy (char x[], char y[])
{    $s0
   int i=0;
   while  ((x[i] = y[i]) != `\0’)
      i += 1;
}
```

```
strcpy:
        addi $sp, $sp, -4
        sw   $s0, 0($sp)
        add  $s0, $zero, $zero
while:  add  $t1, $s0, $a1
        lb   $t2, 0($t1)
        add  $t3, $s0, $a0
        sb   $t2, 0($t3)
        beq  $t2, $zero, exit
        addi $s0, $s0, 1
        j    while
exit:
```

Example: String Copy

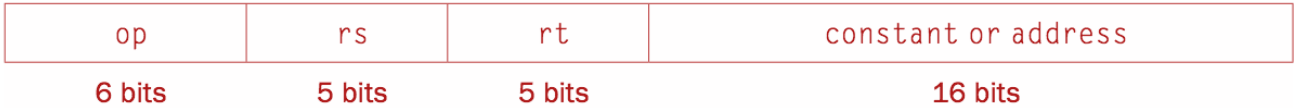
- Convert to Assembly

```
                         $a0      $a1
void strcpy (char x[], char y[])
{     $s0
   int i=0;
   while  ((x[i] = y[i]) != `\0’)
      i += 1;
}
```

```
strcpy:
        addi $sp, $sp, -4
        sw   $s0, 0($sp)
        add  $s0, $zero, $zero
while:  add  $t1, $s0, $a1
        lb   $t2, 0($t1)
        add  $t3, $s0, $a0
        sb   $t2, 0($t3)
        beq  $t2, $zero, exit
        addi $s0, $s0, 1
        j    while
exit:   lw   $s0, 0($sp)
        addi $sp, $sp, 4
        jr   $ra
```

Large Constants

- Immediate instructions can only specify 16-bit constants
  - Recall: I-Type



op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- The `lui` instruction is used to store a 16-bit constant into the upper 16 bits of a register… combine this with an OR instruction to specify a 32-bit constant
  - `lui   $t0, 9`
  - `ori   $a0, $t0, 64497`

- The destination PC-address in a conditional branch is specified as a 16-bit constant, relative to the current PC

- A jump (`j`) instruction can specify a 26-bit constant; if more bits are required, the jump-register (`jr`) instruction is used

Example: Sort Algorithm

- Convert to assembly

```
void sort (int v[ ], int n)
{
    int i, j;
    for (i=0; i<n; i+=1) {
        for (j=i-1; j>=0 && v[j] <= v[j+1]; j-=1) {
            swap (v,j);
        }
    }
}
```

```
void swap (int v[ ], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

(1) Allocate registers to program variables
(2) Produce code for the program body
(3) Preserve registers across procedure invocations

Example: Sort Algorithm

- Convert to assembly

$a0 $a1

```
void swap (int v[ ], int k)
{       $t0
   int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Callee saves $s0-$s7

Example: Sort Algorithm

- Convert to assembly

```
swap: sll  $t1, $a1, 2
      add  $t1, $a0, $t1
      lw   $t0, 0($t1)
      lw   $t2, 4($t1)
      sw   $t2, 0($t1)
      sw   $t0, 4($t1)
      jr   $ra
```

```
                        $a0       $a1
void swap (int v[ ], int k)
{         $t0
   int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

No need for saves and restores as we're not using $s0-$s7
No need to re-use $a0 and $a1)

Example: Sort Algorithm

- Convert to assembly

```
                 $a0       $a1
void sort (int v[ ], int n)
{   $s0, $s1
    int i, j;
    for (i=0; i<n; i+=1) {
        for (j=i-1; j>=0 && v[j] <= v[j+1]; j-=1) {
            swap (v,j);
        }
    }
}
```

Example: Sort Algorithm

- Convert to assembly

```
                $a0       $a1
void sort (int v[ ], int n)
{  $s0, $s1
   int i, j;
   for (i=0; i<n; i+=1) {
      for (j=i-1; j>=0 && v[j] <= v[j+1]; j-=1) {
         swap (v,j);
      }
   }
}
```

```
           move $s0, $zero
loopbody1: bge  $s0, $a1, exit1
      … body of inner loop …
           addi $s0, $s0, 1
           j    loopbody1
exit1:
```

Need to store $a0 and $a1
Note the use of pseudo-instructions

Example: Sort Algorithm

- Convert to assembly

```
for (j=i-1; j>=0 && v[j] <= v[j+1]; j-=1) {
     swap (v,j);
}
```

```
            addi $s1, $s0, -1
loopbody2:  blt  $s1, $zero, exit2
            sll  $t1, $s1, 2
            add  $t2, $a0, $t1
            lw   $t3, 0($t2)
            lw   $t4, 4($t2)
            bgt  $t3, $t4, exit2
      … body of inner loop …
            addi $s1, $s1, -1
            j    loopbody2
exit2:
```

Example: Sort Algorithm

- □ Convert to assembly

```
                 $a0        $a1
void sort (int v[ ], int n)
{  $s0, $s1
   int i, j;
   for (i=0; i<n; i+=1) {
      for (j=i-1; j>=0 && v[j] <= v[j+1]; j-=1) {
           swap (v,j);
      }
   }
}
```

```
                 $a0        $a1
void swap (int v[ ], int k)
{         $t0
   int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Use $s2 and $s3 instead of $a0 and $a1 in the rest of "sort"
Save $ra at the start of "sort"
Save $s0-$s3 so "sort" does not overwrite something that belongs to its caller

Example: Sort Algorithm

□ Saves and restores

```
sort:   addi   $sp, $sp, -20
        sw     $ra, 16($sp)
        sw     $s3, 12($sp)
        sw     $s2, 8($sp)
        sw     $s1, 4($sp)
        sw     $s0, 0($sp)
        move   $s2, $a0
        move   $s3, $a1
             …
        move   $a0, $s2         # the inner loop body starts here
        move   $a1, $s1
        jal    swap
             …
exit1:  lw        $s0, 0($sp)
              …
        addi    $sp, $sp, 20
        jr      $ra
```