

CSE 2021

Computer Organization

Hugh Chesser, CSEB
1012U



Example from last time....



Activity 2: Consider the C instruction

$$A[300] = h + A[300]$$

- A. Write the equivalent MIPS code for the above C instruction assuming \$t1 contains the base address of array A (i.e., address of A[0]) and \$s2 contains the value of h
- B. Write the binary machine language code for the result in part A.

MIPS Code:

```
lw $t2,1200($t1)      # load A[300] from memory
add $t2,$t2,$s2       # A[300]= A[300] + h
sw $t2,1200($t1)     # save result to memory A[300]
```



Example from last time (cont'd)...

Example: `lw $t2,1200($t1)`

I-Format			
op	rs	rt	address
6 bits	5 bits	5 bits	16 bits
opcode	1 st operand	2 nd operand	Memory address (offset)

$$\text{op} = 0x23 = (100011)_2$$

$$\text{rs} = 9 = (01001)_2 \text{ (\$t1 is register 9)}$$

$$\text{rt} = 10 = (01010)_2 \text{ (\$t2 is register 10)}$$

$$\text{address} = 1200 = 0x4b0 = (0000\ 0100\ 1011\ 0000)_2$$

leads to the binary machine language code: **100011 01001 01010 0000 0100 1011 0000**



Completing the Example...

add
\$t2,\$t2,\$s2

R-Format					
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
opcode	1 st operand	2 nd operand	destination	shift	function

$$\text{op} = 0x0 = (000000)_2$$

$$\text{funct} = 0x20$$

$$\text{rs} = 10 = (01010)_2 \text{ (\$t2 is register 10)}$$

$$\text{rt} = 18 = (10010)_2 \text{ (\$s2 is register 18)}$$

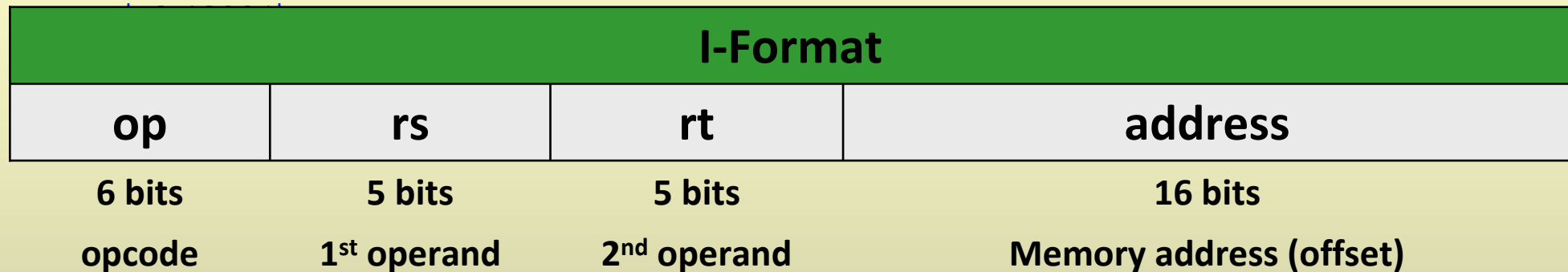
$$\text{rd} = 10 = (01010)_2 \text{ (\$t2 is register 10)}$$

leads to the binary machine language code: **000000 01010 10010 01010 00000 100000**



Completing the Example (cont'd)...

SW



$$\text{op} = 0x2b = (101011)_2$$

$$\text{rs} = 10 = (01001)_2 \text{ (\$t1 is register 9)}$$

$$\text{rt} = 10 = (01010)_2 \text{ (\$t2 is register 10)}$$

$$\text{address} = 1200 = 0x4b0 = (0000\ 0100\ 1011\ 0000)_2$$

leads to the binary machine language code: **101011 01001 01010 0000 0100 1011 0000**



Agenda for Today

- Procedures
- Characters and Strings
- Immediate Operands
- Decoding Machine Language (read on your own)
- Putting it all together (read on your own)
- SPIM Programming (Lab B, C, D)

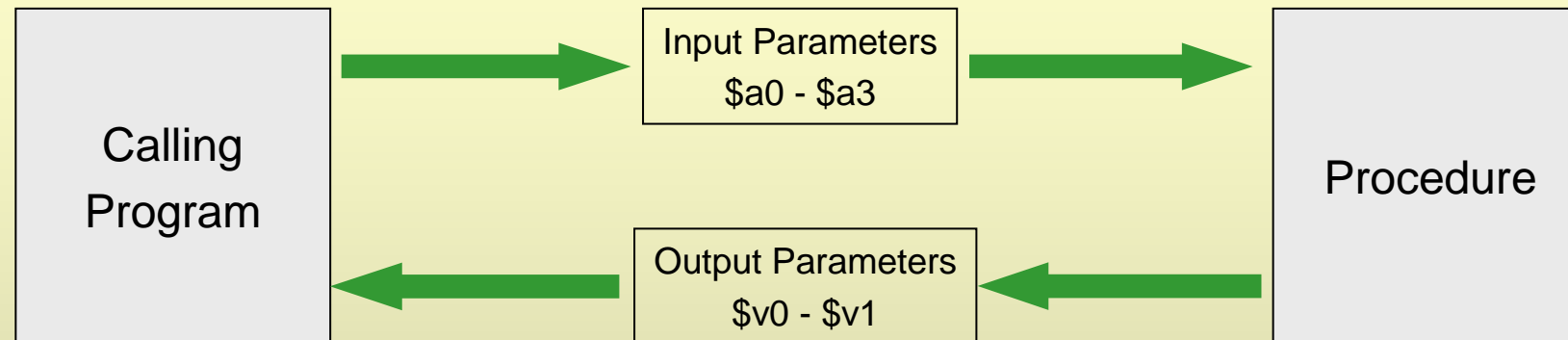
- Patterson: Sections 2.8 – 2.13 and Appendix B.9

What we have learned so far ...



Category	Instruction	Example	Meaning	Comments
Arithmetic	add	<code>add \$s1,\$s2,\$s3</code>	$\$s1 \leftarrow \$s2 + \$s3$	
	subtract	<code>sub \$s1,\$s2,\$s3</code>	$\$s1 \leftarrow \$s2 - \$s3$	
Data Transfer	load word	<code>lw \$s1,100(\$s2)</code>	$\$s1 \leftarrow \text{Mem}[\$s2 + 100]$	
	store word	<code>sw \$s1,100(\$s2)</code>	$\text{Mem}[\$s2 + 100] \leftarrow \$s1$	
Conditional branch	branch on equal	<code>beq \$s1,\$s2,L</code>	$\text{if}(\$s1 == \$s2) \text{ go to } L$	
	branch not equal	<code>bne \$s1,\$s2,L</code>	$\text{if}(\$s1 \neq \$s2) \text{ go to } L$	
	set on less than	<code>slt \$s1,\$s2,\$s3</code>	$\text{if}(\$s2 < \$s3) \$s1 = 1$ $\text{else } \$s1 = 0$	
Unconditional jump	jump	<code>j 2500</code>	go to (4 x 2500)	
	Jump register	<code>jr \$t1</code>	go to \$t1	
Logical Instructions	Shift left (right)	<code>sll \$t2,\$s0,4</code>	$\$t2 \leftarrow \$s0 \ll 4 \text{ bits}$	
	And, Or, Nor	<code>nor \$t0,\$t1,\$t2</code>	$\$t0 = \sim(\$t1 \$t2)$	

Procedures (1)



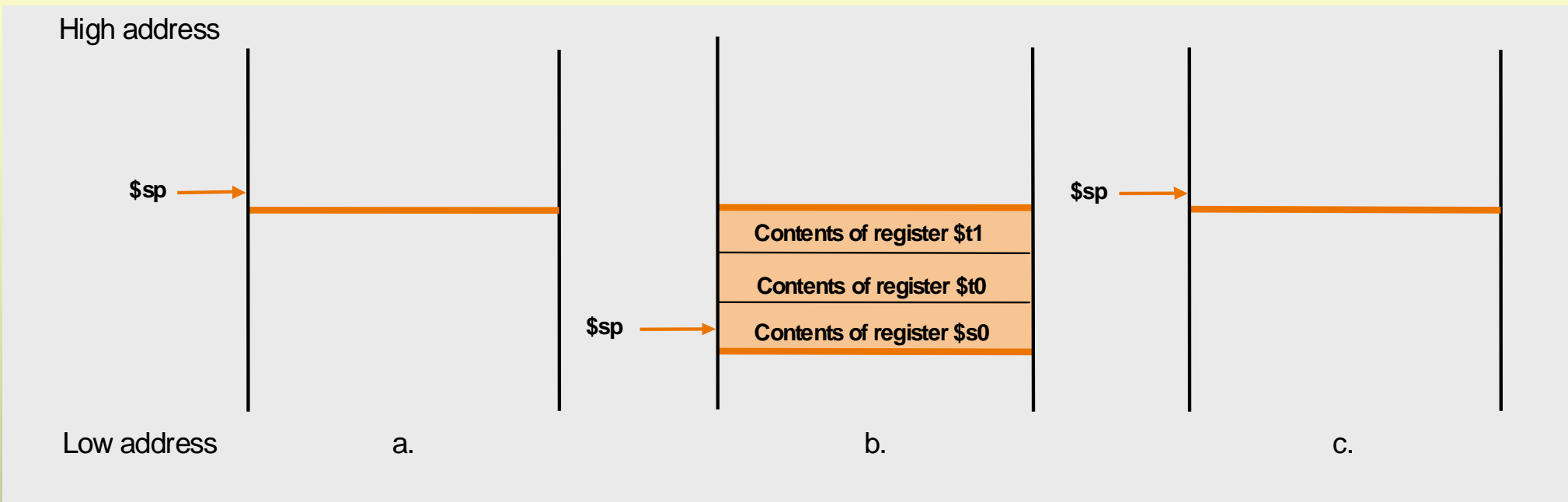
Calling program:

1. Places parameters in registers $\$a0 - \$a3$ where the called procedure can access them
2. Transfers control to the procedure
 - Address of the next instruction in the calling program is automatically placed in register $\$ra$

Called Procedure:

3. Acquires additional space needed to perform the task
 - Saves values of required registers in a stack $\$sp$
4. Performs the desired task
5. Restores the values of registers that it used
6. Saves the result in registers $\$v0 - \$v1$
7. Returns control to the calling program by returning to instruction whose address is saved in $\$ra$

Procedures (2)



Where are values of the calling function stored:

1. The values of registers that you will use in the called function must be stored
2. It is the responsibility of the called function to store these values and restore them
3. A stack (assigned space) in the memory is used to store the values of the registers
4. Register \$sp contain the address of the stack pointer
5. Registers are typically stored in order, from up to down in the stack

Procedures (2)

Example:

```
int leaf_example (int g, int h, int i, int j)
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

Assume that variables g, h, i, j are stored in registers \$a0 - \$a3.

Calling function in MIPS:

```
main:
...
jal leaf_example
...
```

Activity 1:

Why is it perfectly correct not to store and restore registers \$t0, \$t1?

Procedure in MIPS :

leaf_example:

```
# save registers $t0, $t1, $s1
    sub $sp, $sp, 12
    sw $t1, 8($sp)
    sw $t0, 4($sp)
    sw $s0, 0($sp)
```

perform the required operation

```
    add $t0, $a0, $a1
    add $t1, $a2, $a3
    sub $s0, $t0, $t1
    add $v0, $s0, $zero
```

restore registers \$t0, \$t1, \$s1

```
    lw $t1, 8($sp)
    lw $t0, 4($sp)
    lw $s0, 0($sp)
    add $sp, $sp, 12
```

return control

```
    jr $ra
```



Procedures: Nested (3)



Nested Procedure Example:

```
int fact (int n)
{
    if (n < 1) return (1)
    else return (n * fact(n-1));
}
```

Assume variable n is stored in $\$a0$.

Conclusions:

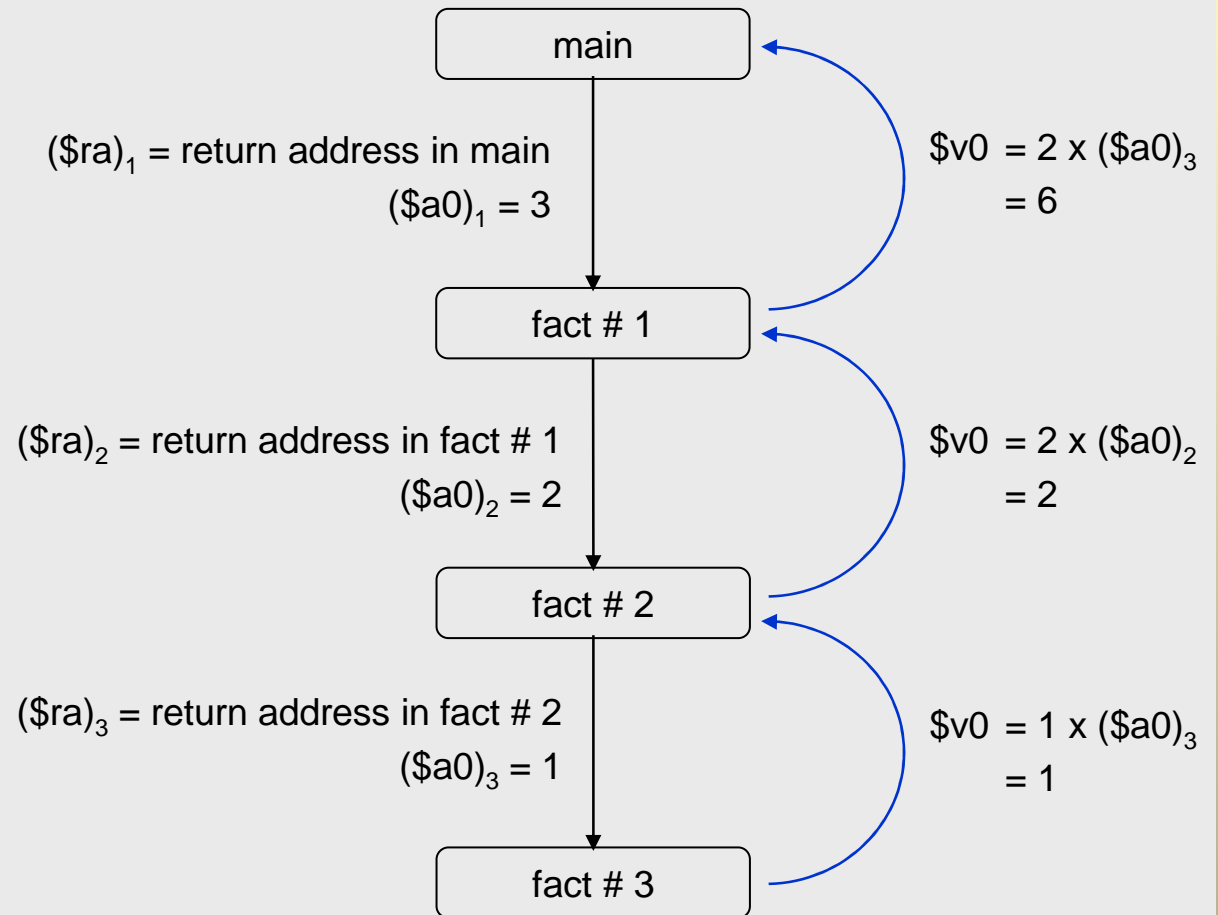
Before each function call

- 1) value of $\$ra$ must be saved in the stack
- 2) value of $\$a0$ must be saved in the stack

After returning from a function

- 3) value of $\$a0$ be recalled for $(\$v0 \times \$a0)$
- 4) value of $\$ra$ be recalled to return control to the calling function

Flow Diagram



Procedures: Nested (4)



Nested Procedure Example:

```
int fact (int n)
{
    if (n < 1) return (1)
    else return (n * fact(n-1));
}
```

Assume variable `n` is stored in `$a0`.

MIPS Code:

fact:

```
# input value is stored in $a0 and output value in $v0
# add commands for saving $ra, $a0
```

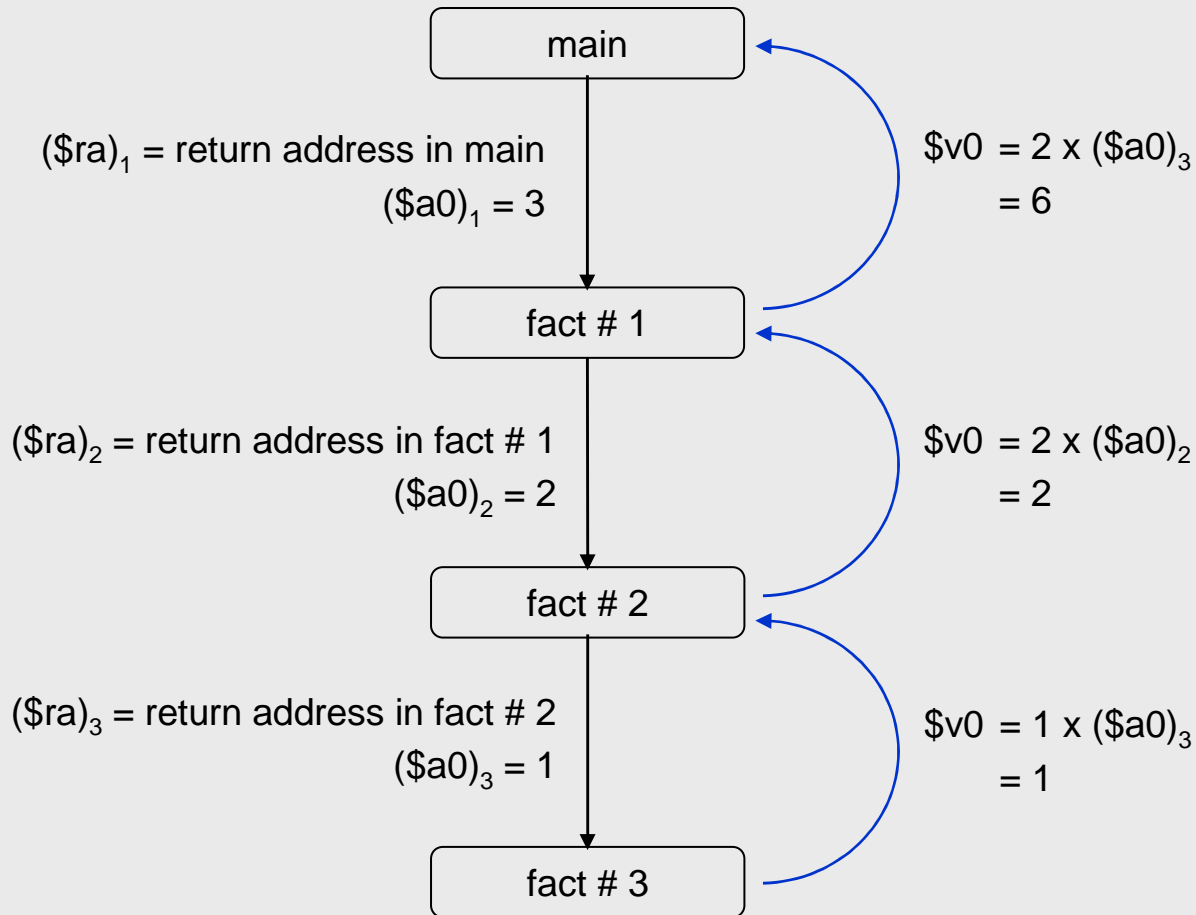
```
addi    $t1, $zero, 1    # initialize $t1 = 1
slt     $t0, $a0, $t1    # if (n < 1), $t0 = 1
beq     $t0, $zero, L1   # if $t0 == 0, go to L1
# if n < 1
addi    $v0, $zero, 1    # return 1
jr      $ra
# if n >= 1
L1: addi    $a0, $a0, -1    # $a0 = $a0 - 1
jal fact
```

```
# add commands to retrieve $ra, $a0
# return answer in $v0
```

Procedures: Nested (5)



Flow Diagram



MIPS Code:

fact:

```

addi $sp, $sp, -8    # clear stack for 2 items
sw $ra, 4($sp)      # save return address
sw $a0, 0($sp)      # save n
  
```

```

addi $t1, $zero, 1  # initialize $t1 = 1
slt $t0, $a0, $t1   # if (n < 1), $t0 = 1
beq $t0, $zero, L1  # if $t0 == 0, go to L1
  
```

if n < 1

```
add $v0, $zero, 1    # return 1
```

```
addi $sp, $sp, 8
```

```
jr $ra
```

if n >= 1

```
L1: addi $a0, $a0, -1 # $a0 = $a0 - 1
```

```
jal fact
```

```
lw $a0, 0($sp)
```

```
lw $ra, 4($sp)
```

```
addi $sp, $sp, -8
```

```
mul $v0, $a0, $v0    # not exact instruction
```

```
jr $ra
```

Registers (review)



Name	Example	Comments
32 Registers	<code>\$s0, \$s1, ... \$s7, \$zero</code> <code>\$t0, \$t1, ... \$t9</code>	\$s0-\$s7 are preserved across procedures, \$t0-\$t9 are not preserved
Memory w/ 2^{30} words	Memory[0], Memory[4], ... Memory[4294967292]	Memory is accessed one word at a time

Name	Register number	Usage
<code>\$zero</code>	0	Constant value of 0
<code>\$v0-\$v1</code>	2 - 3	Values for results and expression evaluation
<code>\$a0-\$a3</code>	4 - 7	Input arguments to a procedure
<code>\$t0-\$t7</code>	8 - 15	Not preserved across procedures (temp)
<code>\$s0-\$s7</code>	16 - 23	Preserved across procedure calls
<code>\$t8-\$t9</code>	24 - 25	More temporary registers
<code>\$gp</code>	28	Global pointer
<code>\$sp</code>	29	Stack pointer, points to last location of stack
<code>\$fp</code>	30	Frame pointer
<code>\$ra</code>	31	Return address from a procedure call

ASCII Characters (1)



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

ASCII Characters (2)



1. Characters are represented on a computer using **American Standard Code for Information Interchange (ASCII)**. Java uses **Unicode**.
2. Most computers use 8-bits to represent each character (16-bits for Unicode)
Example: ASCII code for character **C** is $(67)_{10}$ or $(01000011)_2$
 ASCII code for character **null** is $(\backslash 0)_{10}$ or $(00000000)_2$
3. Strings are combination of characters
4. In ANSI C, the end of a string is represented by a null character

	72	101	108	108	111	32	83	105	114	0
--	----	-----	-----	-----	-----	----	----	-----	-----	---

5. Other high level languages may use other choices as:
 - a. Using the first byte to represent the length of the string
 - b. Using an accompanying variable to indicate the length of the string
6. How to load a byte at a time? (Unicode has half-byte instructions lh, lhu, sh)

```
lb $t0,100($s1) # $t0 = Mem[$s1 + 100] except 1 byte is transferred  
sb $t0,100($s1) # Mem[$s1 + 100] = $t0 except 1 byte is transferred
```


ASCII Characters (3)



Example:

```
void strcpy (char x[ ], char y [ ])  
{  
    int j;  
    j = 0;  
    while ((x[j] = y[j] != 0) /* copy and test */  
           i = i + 1;  
}
```

Assume that the addresses of x[0] and y[0] are contained in registers \$a0, \$a1.

strcpy:

```
# store value of register $s0  
addi $sp,$sp,-4  
sw $s0,0($sp)  
  
add $s0,$zero,$zero # set $s0 = 0  
L1: add $t1,$a1,$s0 # address of y[i]  
    lb $t2,0($t1) # load y[i] to $t2  
    add $t3,$a0,$s0 # address of x[i]  
    sb $t2,0($t3) # save $t2 to x[i]  
    beq $t2,$zero,L2 # exit  
    addi $s0,$s0,1  
    j L1  
  
# restore value of register $s0  
L2: lw $s0,0($sp)  
    addi $sp,$sp,4  
    jr $ra
```



Immediate Operands (1)

1. Set value of register to a constant

```
addi $s0,$zero,88    # $s0 = 88
```

Machine code for `addi` instruction:

op	rs	rt	Immediate
$8_{10} = 001000_2$	$0_{10} = 00000_2$	$16_{10} = 10000_2$	$88_{10} = 0000\ 0000\ 0101\ 1000_2$
6 bits	5 bits	5 bits	16 bits
opcode	1st operand	2nd operand	Immediate operand

2. What if the value of constant to be loaded is greater than $(2^{16} - 1) = 65535$?

Example: Load the following binary number into \$s0?

```
0000 0000 0011 1101 0000 1001 0000 0000
```

```
lui $s0,61           # load 61 = 0000 0000 0011 1101 into the most
                    # significant 16 bits
```

```
addi $s0,$s0,2304
```



Immediate Operands (2)

1. Branch statements

`bne $s0,$s1,Exit` # go to Exit if \$s0 != \$s1

Machine code for `bne` instruction:

op $5_{10} = 000101_2$	rs $\$s1 = 00000_2$	rt $\$s0 = 10000_2$	Immediate address of Exit
6 bits	5 bits	5 bits	16 bits
opcode	1st operand	2nd operand	Immediate operand

2. What if the value of address corresponding to Exit is greater than $(2^{16} - 1) = 65535$?

Use unconditional jump with syntax: `j 10000`

op $2_{10} = 000010_2$	Immediate		
6 bits	26 bits		
opcode	1st operand	2nd operand	Immediate operand

MIPS Code: `beq $s0,$s1,L2`
`L2: j L1`

SPIM Simulator



1. SPIM is a software simulator for running MIPS program
2. SPIM's name is just MIPS spelled backwards
3. There are different versions for different machines:
 - Unix: spim, xspim
 - PC: PCspim (download instructions available on course homepage)
4. SPIM provides additional features not available in MIPS like system calls
 - Systems calls are operating-system like calls for inputting variables, displaying results, etc.
 - Format for system calls is:

```
place value of input argument in $a0
place value of system-call-code in $v0
syscall
```

System Calls



Example # 1: Print a string

```
.data
    str:    .asciiz "the answer is"
.text
    addi $v0,$zero,4
    la $a0,str    # pseudoinstruction
    syscall
```

Example # 2: Input an integer

```
addi $v0,$zero,5
syscall
```

Example # 3: Print an integer

```
addi $v0,$zero,1
addi $a0,$s0,$zero
syscall
```

Example # 4: Read String

```
addi $v0,$zero,8
la $a0, Buff    # $a0=address of Buff
addi $a1,$zero,60 # $a1=max. len.
syscall
```

Service	System Call Code (\$v0)	Arguments	Result
print_int	1	\$a0 = int	
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string address	
read_int	5		int (in \$v0)
read_float	6		float (in \$f0)
read_double	7		double (in \$f0)
read_string	8	\$a0 = buffer \$a1 = length	
sbrk	9	\$a0 = amount	address (in \$v0)
exit	10		terminate prog

Putting it all together (2)



Activity: Write a MIPS program which does the following:

1. Accepts an integer N using the following prompt
Please input a value for $N =$
2. Computes the sum of integers from 1 to N , i.e., $(1 + 2 + \dots + N)$ if $N > 0$
3. Displays the result (X) as
The sum of the integers from 1 to N is X
4. Waits for the next number N .
5. If $N \leq 0$, the program exits with the following farewell
Chao - Have a good day

Run the program in the spim simulator to verify the results