	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R:	op	rs	rt	rd	shamt	funct
I:	op	rs	rt	address / immediate		
J:	op	target address				

MIPS Addressing
MIPS Instructions

CS 64: Computer Organization and Design Logic

Lecture #8

Winter 2020

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB

This Week on "Didja Know Dat?!"

Steve Wozniak and Steve Job's first commercial venture was the **Apple 1** in **1976** using an **8-bit MOS 6502 CPU**. It was built for $500 and initially **sold for $666.66** because Wozniak "*liked repeating digits*" (about $2900 in today's dollars). Keyboard and TV not included. They sold about 200 of them in 10 months, thus assuring the continuation of their company.

Previously, the only other popular "personal" computer was the Altair 8800, which you had to operate with switches!

Administrative

- Lab 4 due tomorrow!

Lecture Outline

- MIPS Instructions
  - How they are represented
- Overview of Functions in MIPS

Midterm Exam (**Wed. 2/12**)

What's on It?

- Everything we've done so far from start to Monday, 2/10

What Should I Bring?

- Your pencil(s), eraser, MIPS Reference Card (on 1 page)
- THAT'S ALL!

What Else Should I Do?

- ***IMPORTANT***: Come to the classroom 5-10 minutes EARLY
- **If you are late, I may not let you take the exam**
- ***IMPORTANT***: Use the bathroom before the exam – once inside, you cannot leave
- I will have some of you re-seated
- Bring your UCSB ID

Any Questions From Last Lecture?

Let's review the array exercise…

```
.data
newline: .asciiz "\n"
myArray: .word 5 32 87 95 286 386
myArrayLength: .word 6

.text
main:
	# t0: x
	# initialize x
	li $t0, 0
	# get myArrayLength, put result in $t2
	# $t1 = &myArrayLength
	la $t1, myArrayLength
	lw $t2, 0($t1)
```

```
int myArray[]
        = {5, 32, 87, 95, 286, 386};
int myArrayLength = 6;
int x;

for (x = 0; x < myArrayLength; x++)
{
    print(myArray[x]);
    print("\n");
}
```

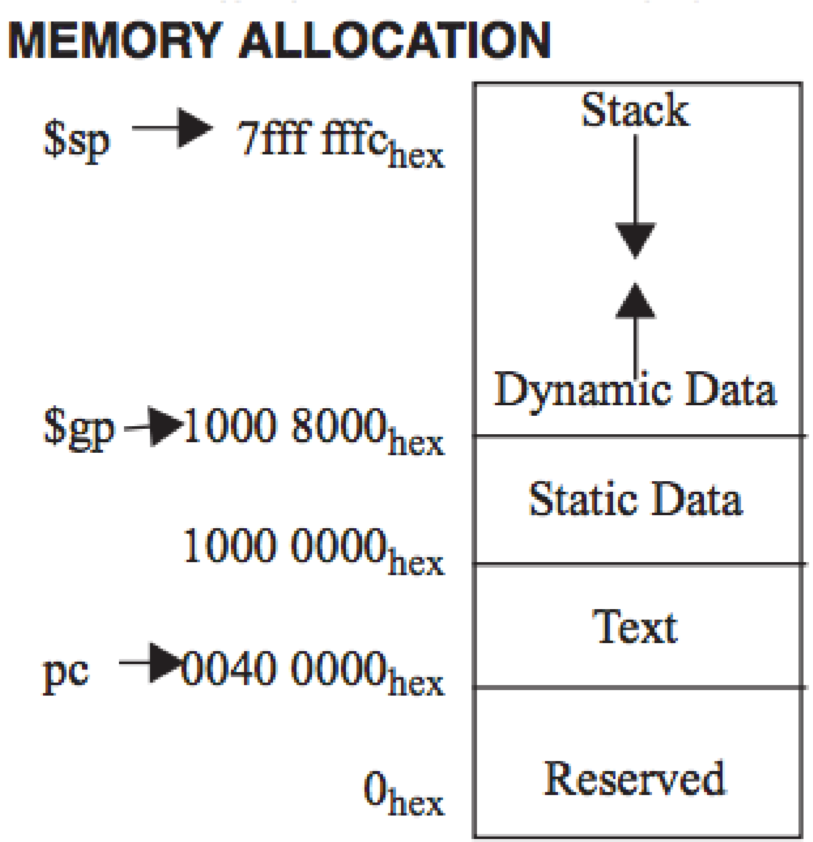
```
loop:
	# see if x < myArrayLength
	# put result in $t3
	slt $t3, $t0, $t2

	# jump out if not true
	beq $t3, $zero, end_main
```

```
# get the base of myArray
la $t4, myArray

# figure out where in the array we need
# to read from. This is going to be the array
# address + (index << 2). The shift is a
# multiplication by four to index bytes
# as opposed to words.
# Ultimately, the result is put in $t7
sll $t5, $t0, 2
add $t6, $t5, $t4
lw $t7, 0($t6)
```

```
    # print x[i] out, with a newline
    li $v0, 1
    move $a0, $t7
    syscall
    li $v0, 4
    la $a0, newline
    syscall

    # increment index
    addi $t0, $t0, 1

    # restart loop
    j loop

end_main:
    # exit the program
    li $v0, 10
    syscall
```

Memory Allocation Map

How much memory does a programmer get to directly use in MIPS?



This is found on your ***MIPS Reference Card***

NOTE:

Not all memory addresses can be accessed by the programmer.

Although the address space is 32 bits, the top addresses from **0x80000000** to **0xFFFFFFFF** are not available to user programs. They are used mostly by the OS.

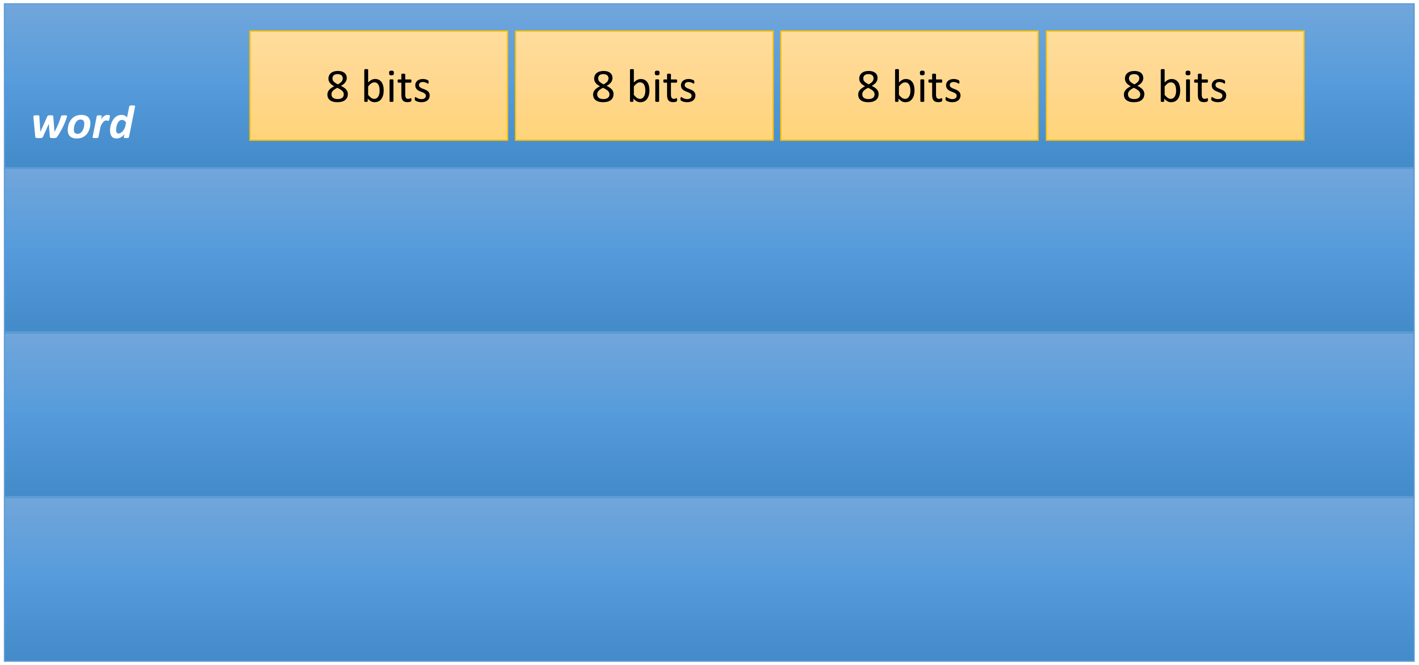
Mapping MIPS Memory

(say that 10 times fast!)

- Imagine computer memory like a big array of words
- Size of computer memory is:

  2^{32} = 4 Gbits, or 512 MBytes (MB)

  - We only get to use 2 Gbits, or 256 MB
  - That's (256 MB/ groups of 4 B) = 64 million words



MIPS Computer Memory Addressing Conventions

	1A	80	C5	29
	0x0000	0x0001	0x0002	0x0003
	52	00	37	EE
A	0x0004	0x0005	0x0006	0x0007
→	B1	11	1A	A5
	0x0008	0x0009	0x000A	0x000B

MIPS Computer Memory Addressing Conventions

or...

B ←

1A	**80**	**C5**	**29**
0x0003	0x0002	0x0001	0x0000
52	**00**	**37**	**EE**
0x0007	0x0006	0x0005	0x0004
B1	**11**	**1A**	**A5**
0x000B	0x000A	0x0009	0x0008

A Tale of 2 Conventions…

BIG END (MSByte) gets addressed first

← BIG ENDIAN

1A	80	C5	29
0x0000	0x0001	0x0002	0x0003
52	00	37	EE
0x0004	0x0005	0x0006	0x0007
B1	11	1A	[illegible]
0x0008	0x0009	0x000[illegible]	[illegible]

LITTLE END (LSByte) gets addressed first

LITTLE ENDIAN →

1A	80	C5	29
0x0003	0x0002	0x0001	0x0000
[illegible]2	00	37	EE
[illegible]007	0x0006	0x0005	0x0004
B1	11	1A	A5
0x000B	0x000A	0x0009	0x0008

The Use of Big Endian vs. Little Endian

Origin: Jonathan Swift (author) in "Gulliver's Travels".

Some people preferred to eat their hard boiled eggs from the "little end" first (thus, little endians), while others prefer to eat from the "big end" (i.e. big endians).

- MIPS users typically go with Big Endian convention
  - MIPS allows you to program "endian-ness"
- Most Intel processors go with Little Endian…
- It's just a convention – it makes no difference to a CPU!

MIPS Reference Card

- Let's take another close look at that card…

Instruction Representation

Recall: A MIPS instruction has 32 bits

32 bits are divided up into 6 fields *(aka the* ***R-Type*** *format)*

• **op** code	6 bits	basic operation
• **rs** code	5 bits	first register source operand
• **rt** code	5 bits	second register source operand
• **rd** code	5 bits	register destination operand
• **shamt** code	5 bits	shift amount
• **funct** code	6 bits	function code

Why did the designers allocate 5 bits for registers?

op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 b	5 b	5 b	5 b	5 b	6 b
31 – 26	*25 – 21*	*20 – 16*	*15 – 11*	*10 – 6*	*5 – 0*

Instruction Representation in R-Type

op	rs	rt	rd	shamt	funct
6 b	5 b	5 b	5 b	5 b	6 b
31 – 26	*25 – 21*	*20 – 16*	*15 – 11*	*10 – 6*	*5 – 0*

- The combination of the **opcode** and the **funct** code tell the processor what it is supposed to be doing
- Example:

```
add $t0, $s1, $s2
```

op	rs	rt	rd	shamt	funct
0	17	18	8	0	32

op = 0, funct = 32	mean “add”
rs = 17	means “$s1”
rt = 18	means “$s2”
rd = 8	means “$t0”
shamt = 0	means this field is unused in this instruction

A full list of codes can be found in your ***MIPS Reference Card***

Exercises

- Using your MIPS Reference Card, write the 32 bit instruction (using the R-Type format and decimal numbers for all the fields) for the following:

```
add $t3, $t2, $s0        0x01505820
addu $a0, $a3, $t0       0x00E82021
sub $t1, $t1, $t2        0x012A4822
```

Exercise: Example Run-Through

- Using your MIPS Reference Card, write the 32 bit instruction (using the R-Type format) for the following. Express your final answer in hexadecimal.

```
add $t3, $t2, $s0    0x01505820
```

op (6b)	**rs** (5b)	**rt** (5b)	**rd** (5b)	**shamt** (5b)	**funct** (6b)
0	10	16	11	0	32
000000	0 1010	1 0000	0 1011	0 0000	10 0000
00000001010100000101100000100000					
0x01505820					

A Second Type of Format…

32 bits are divided up into 4 fields *(the **I-Type** format)*

- **op** code 6 bits basic operation
- **rs** code 5 bits first register source operand
- **rt** code 5 bits second register source operand
- **address** code 16 bits constant or memory address

Note: The I-Type format uses the ***address*** field to access $\pm 2^{15}$ addresses from whatever value is in the ***rs*** field

op	rs	rt	address
6 b	5 b	5 b	16 b
31 – 26	*25 – 21*	*20 – 16*	*15 – 0*

I-Type Format

op	rs	rt	address
6 b	5 b	5 b	16 b
31 – 26	*25 – 21*	*20 – 16*	*15 – 0*

- The I-Type ***address*** field is a <u>signed</u> number

- The **addi** instruction is an I-Type, example:

```
addi $t0, $t1, 42
```

  - What is the largest, most positive, number you can put as an immediate?

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT
Add	add	R
Add Immediate	addi	I
Add Imm. Unsigned	addiu	I
Add Unsigned	addu	R
And	and	R
And Immediate	andi	I
Branch On Equal	beq	I
Branch On Not Equal	bne	I
Jump	j	J
Jump And Link	jal	J
Jump Register	jr	R
Load Byte Unsigned	lbu	I
Load Halfword Unsigned	lhu	I
Load Linked	ll	I
Load Upper Imm.	lui	I
Load Word	lw	I
Nor	nor	R
Or	or	R
Or Immediate	ori	I
Set Less Than	slt	R
Set Less Than Imm.	slti	I
Set Less Than Imm. Unsigned	sltiu	I
Set Less Than Unsig.	sltu	R
Shift Left Logical	sll	R
Shift Right Logical	srl	R
Store Byte	sb	I
Store Conditional	sc	I
Store Halfword	sh	I
Store Word	sw	I
Subtract	sub	R
Subtract Unsigned	subu	R

Ans: 2^{15} - 1

Instruction Representation in I-Type

op	rs	rt	address
6 b	5 b	5 b	16 b
31 – 26	*25 – 21*	*20 – 16*	*15 – 0*

- Example:

addi $t0, $s0, 124

op	**rs**	**rt**	**address/const**
8	16	8	124

op = 8	mean "addi"
rs = 16	means "$s0"
rt = 8	means "$t0"
address/const = 124	is the immediate value

A full list of codes can be found in your ***MIPS Reference Card***

Exercises

- Using your MIPS Reference Card, write the 32 bit instruction (using the I-Type format and decimal numbers for all the fields) for the following:

```
addi $t3, $t2, -42      0x214BFFD6
andi $a0, $a3, 1        0x30E40001
slti $t8, $t8, 14       0x2B18000E
```

YOUR TO-DOs

- Do readings!
  - Check syllabus for details!

- Turn in Assignment #4

</LECTURE>