

MIPS Architecture

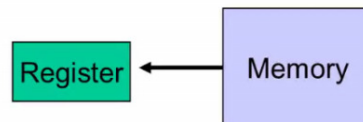
- ❑ Multiplication and division instructions, which run asynchronously from other instructions.
- ❑ A pair of 32-bit registers, **HI** and **LO**, are provided.
- ❑ The program counter has 32 bits.
- ❑ The two low-order bits always contain zero. Why?
- ❑ MIPS I instructions are 32 bits long and are aligned to their natural word boundaries.

MIPS Architecture

- ❑ Values must be fetched from memory before (add and sub) instructions can operate on them

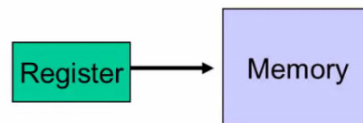
Load word:

lw \$t0, memory-address



Store word:

sw \$t0, memory-address



MIPS Architecture – Memory Organization

- ❑ Viewed as a large single-dimension array with access by address
- ❑ A memory address is an index into the memory array
- ❑ Byte addressing means that the index points to a byte of memory, and that the unit of memory accessed by a load/store is a byte
- ❑ Bytes are load/store units, but most data items use larger words
- ❑ For MIPS, a word is 32 bits or 4 bytes.
- ❑ 2^{32} bytes with byte addresses from 0 to $2^{32}-1$
- ❑ 2^{30} words with byte addresses 0, 4, 8, ... $2^{32}-4$
 - ❑ words are aligned
 - ❑ what are the least 2 significant bits of a word address?

0	8 bits of data
1	8 bits of data
2	8 bits of data
3	8 bits of data
4	8 bits of data
5	8 bits of data
6	8 bits of data

...

Meet - dwh-bfqi-psf - Mozilla Firefox

Search results - Google | Notable - The Markdown | Slides - Google Drive | orphan process - Google | WhatsApp | Meet - dwh-bfqi-psf

REC | Suvadip Batabyal is presenting | IRA INDRANIL BHO... and 108 more | You

MIPS Architecture

- \$gp points to the area in memory that saves global variables.
- The space allocated on stack by a procedure is termed the activation record (includes saved values and data local to the procedure).

SpL

block - 64K *2³²* *2¹⁶* *64K*

- Frame pointer points to the start of the record. *\$gp*
- Stack pointer points to the end.
- Variable addresses are specified relative to \$fp as \$sp may change during the execution of the procedure
- Dynamically allocated storage (with malloc()) is placed on the heap

RUPSA DHAR has left the meeting

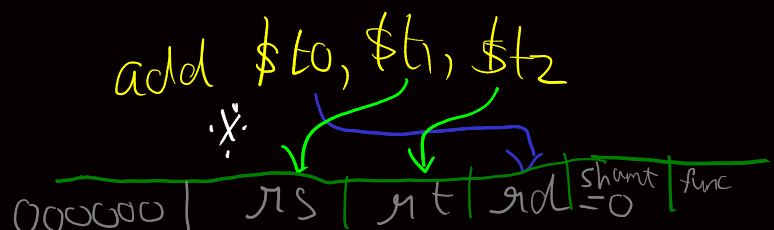
R | *I* | *J* formats
Regs | *immediate* | *jump*

Type

type						
R	opcode(6)	rs(5)	rt(5)	rd(5)	shamt(5)	funct(6)
<u>I</u>	opcode(6)	rs(5)	immediate (16)			
<u>J</u>	opcode(6)	address (26)				

rs - register source | *rt* - reg. target | *rd* = reg. destination
shamt = Shift amount | *funct* - function field (select variant)
 (function code #)

All R type instructions have 00000 as opcode
 Operand order is fixed
 C-Code *a = b + c;*



add - R-type
addi = I-type

add - R-type
addi = I-type

add: \$17, \$2, 1



T type

→ Pad the end with 00 (alignment) each instruction is word aligned

→ Pad the end with 00 (alignment) each instruction is word aligned

→ The first 4 are the ^{first 4 bits of} PC → why (?) why specifically PC? form of segmentation

why specifically PC?
form of segmentation

4(\$2) - 4 bytes offset from value in \$2 (256 megs)

→ (goes to stmt i think for R-type)

32 bit immediate value in a register \rightarrow

li $\neq 10, 0x12345678$

```
lui $t0, 0x1234 -> load upper immediate
ori $t0, 0x5678
```

conditional jumps - 16 bits -> program must fit into 16 bits
Use PC-Relative then

bne is at 80012 , Exit is at 800024
so the address is supposed to be $12/4 = 3$

jump is unconditional however -> it's jumping to 80000 -> address is $80000/4 = 20000$

$\frac{12}{4} = 3$
 80000
 Loop: \$t1, \$s3, \$t2
 add \$t1, \$t1, \$s6
 lw \$t0, 0(\$t1)
 bne \$t0, \$s5, Exit
 addi \$s3, \$s3, 1
 Loop
 Exit:

PC → 80012
 $\frac{12}{4} = 3$
 80000
 Loop: \$t1, \$s3, \$t2
 add \$t1, \$t1, \$s6
 lw \$t0, 0(\$t1)
 bne \$t0, \$s5, Exit
 addi \$s3, \$s3, 1
 Loop
 Exit:

Assumption: program starts at address 80000 in memory.

80000	0	0	19	9	2	0	loop add lw bne addi loop exit
80004	0	9	22	9	0	32	
80008	35	9	8		0		
80012	5	8	21	branch = 2			
80016	8	19	19		1		\rightarrow 5-type exit
80020	2			2.00000 =			
80024	...						