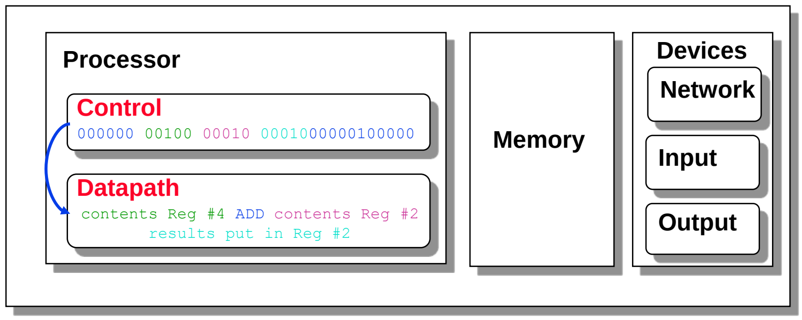
Lecture 2

- Introduction to MIPS assembler, adds/loads/stores

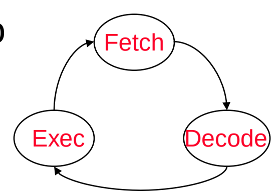
Review: Execute Cycle

The datapath executes the instructions as directed by control



Memory stores both instructions and data

Review: Processor Organization



- Control needs to have circuitry to
  - Decide which is the next instruction and input it from memory
  - Decode the instruction
  - Issue signals that control the way information flows between datapath components
  - Control what operations the datapath's functional units perform
- Datapath needs to have circuitry to
  - Execute instructions - functional units (e.g., adder) and storage locations (e.g., register file)
  - Interconnect the functional units so that the instructions can be executed as required
  - Load data from and store data to memory

Assembly Language Instructions

- The language of the machine
  - Want an ISA that makes it *easy* to build the hardware and the compiler while maximizing performance and minimizing cost
- Stored program concept
  - Instructions are stored in memory (as the data)
- Our target: the MIPS ISA
  - similar to other ISAs developed since the 1980's
  - used by Broadcom, Cisco, NEC, Nintendo, Sony, …

Design goals: maximize performance, minimize cost, reduce design time (time-to-market), minimize memory space (embedded systems), minimize power consumption (mobile systems)

RISC - Reduced Instruction Set Computer

- RISC philosophy
  - fixed instruction lengths
  - load-store instruction sets
  - limited number of addressing modes
  - limited number of operations
- MIPS, Sun SPARC, HP PA-RISC, IBM PowerPC …
- Instruction sets are measured by how well compilers use them as opposed to how well assembly language programmers use them
- CISC (C for complex), e.g., Intel x86

Design Principles

1. Simplicity favors regularity.
2. Smaller is faster.
3. Make the common case fast.

MIPS Arithmetic Instruction

- MIPS assembly language arithmetic statement

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

- Each arithmetic instruction performs only one operation
- Each arithmetic instruction specifies exactly three operands

  destination ← source1 op source2

  - Operand order is fixed (the destination is specified first)
- The operands are contained in the datapath's register file (`$t0, $s1, $s2`)

Compiling More Complex Statements

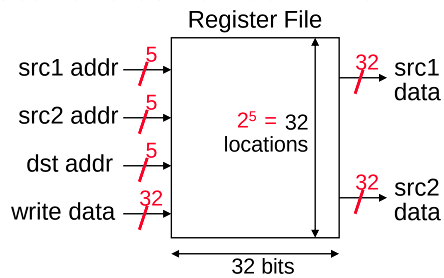
- Assuming variable b is stored in register `$s1`, c is stored in `$s2`, and d is stored in `$s3` and the result is to be left in `$s0`, what is the assembler equivalent to the C statement

```
h = (b - c) + d
```

```
sub  $t0, $s1, $s2
add  $s0, $t0, $s3
```

MIPS Register File

- Operands of arithmetic instructions must be from a limited number of special locations contained in the datapath's register file
  - Thirty-two 32-bit registers
    - Two read ports
    - One write port



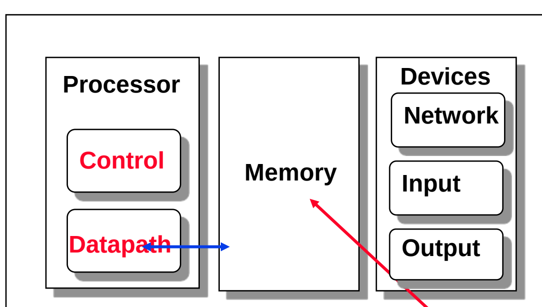
- Registers are
  - Fast
    - Smaller is faster & Make the common case fast
  - Easy for a compiler to use
  - Improves code density
    - Since register are named with fewer bits than a memory location
- Register addresses are indicated by using $

Naming Conventions for Registers

0	$zero	constant 0 (Hdware)	16	$s0	callee saves
1	$at	reserved for assembler	. . .		(caller can clobber)
2	$v0	expression evaluation &	23	$s7	
3	$v1	function results	24	$t8	temporary (cont'd)
4	$a0	arguments	25	$t9	
5	$a1		26	$k0	reserved for OS kernel
6	$a2		27	$k1	
7	$a3		28	$gp	pointer to global area
8	$t0	temporary: caller saves	29	$sp	stack pointer
. . .		(callee can clobber)	30	$fp	frame pointer
15	$t7		31	$ra	return address (Hdware)

Registers vs. Memory

- Arithmetic instructions *operands* must be in registers
  - only thirty-two registers are provided

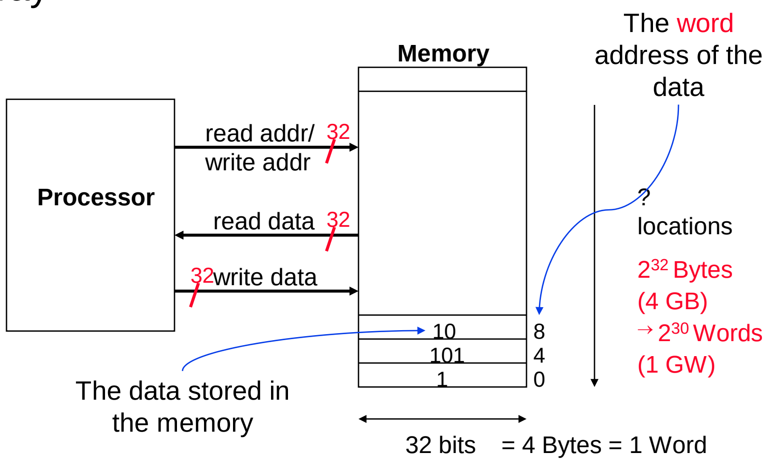


- Compiler associates variables with registers

What about programs with lots of variables?

Processor – Memory Interconnections

- Memory is a large, single-dimensional array
- An address acts as the index into the memory array



Accessing Memory

- MIPS has two basic data transfer instructions for accessing memory (assume `$s3` holds 24_{10})

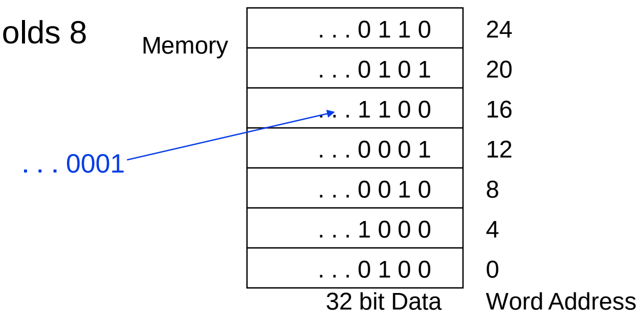
```
lw    $t0, 4($s3)   #load word from memory
             28
sw    $t0, 8($s3)   #store word to  memory
             32
```

- The data transfer instruction must specify
  - where in memory to read from (load) or write to (store) – memory address
  - where in the register file to write to (load) or read from (store) – register destination (source)
- The memory address is formed by summing the constant portion of the instruction and the contents of the second register

MIPS Memory Addressing

- The memory address is formed by summing the constant portion of the instruction and the contents of the second (base) register

`$s3` holds 8



. . . 0001

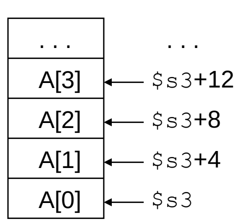
```
lw    $t0, 4($s3)   #what? is loaded into $t0
sw    $t0, 8($s3)   #$t0 is stored where?
```

in location 16

Compiling with Loads and Stores

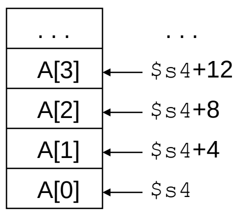
- Assuming variable b is stored in `$s2` and that the base address of array A is in `$s3`, what is the MIPS assembly code for the C statement

`A[8] = A[2] - b`




```
lw    $t0, 8($s3)
sub   $t0, $t0, $s2
sw    $t0, 32($s3)
```

Compiling with a Variable Array Index



- Assuming that the base address of array A is in register `$s4`, and variables b, c, and i are in `$s1`, `$s2`, and `$s3`, respectively, what is the MIPS assembly code for the C statement

  `c = A[i] - b`

```
add   $t1, $s3, $s3      #array index i is in $s3
add   $t1, $t1, $t1      #temp reg $t1 holds 4*i
add   $t1, $t1, $s4      #addr of A[i] now in $t1
lw    $t0, 0($t1)
sub   $s2, $t0, $s1
```

Dealing with Constants

- Small constants are used quite frequently (50% of operands in many common programs)

  e.g., A = A + 5;
  B = B + 1;
  C = C - 18;

Constant (or Immediate) Operands

- Include constants inside arithmetic instructions
  - Much faster than if they have to be loaded from memory (they come in from memory *with* the instruction itself)

- MIPS immediate instructions

```
addi $s3, $s3, 4  #$s3 = $s3 + 4
```

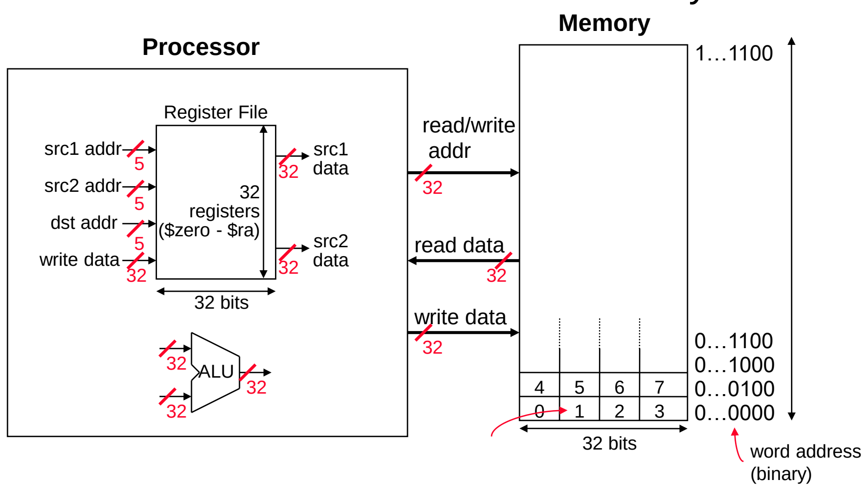
There is no `subi` *instruction, can you guess why not?*

MIPS Instructions, so far

Category	Instr	Example	Meaning
Arithmetic	add	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	sub $s1, $s2, $s3	$s1 = $s2 - $s3
	add immediate	addi $s1, $s2, 4	$s1 = $s2 + 4
Data transfer	load word	lw $s1, 32($s2)	$s1 = Memory($s2+32)
	store word	sw $s1, 32($s2)	Memory($s2+32) = $s1

Review: MIPS Organization

- Arithmetic instructions – to/from the register file
- Load/store instructions - to/from memory

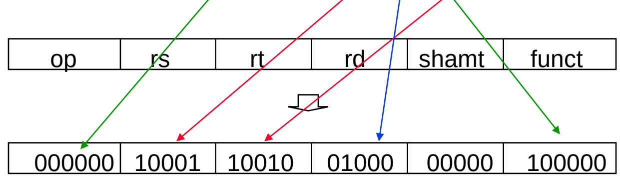


Machine Language - Arithmetic Instruction

- Instructions, like registers and words of data, are also 32 bits long
  - Example: add $t0, $s1, $s2

  registers have numbers $t0=$8, $s1=$17, $s2=$18

- Instruction Format:



Can you guess what the field names stand for?

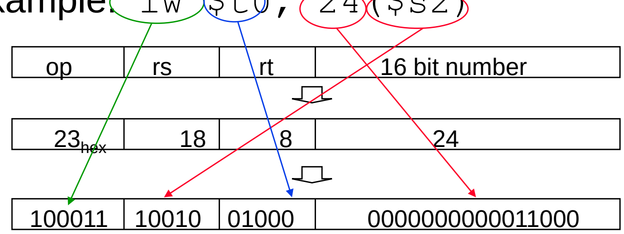
MIPS Instruction Fields

op	rs	rt	rd	shamt	funct	
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	= 32 bits

- *op*   **op**code indicating operation to be performed
- *rs*   address of the first **r**egi**s**ter source operand
- *rt*   address of the second **r**egis**t**er source operand
- *rd*   the **r**egister **d**estination address
- *shamt*   **sh**ift **am**oun**t** (for shift instructions)
- *funct*   **funct**ion code that selects the specific variant of the operation specified in the opcode field

Machine Language - Load Instruction

- Consider the load-word and store-word instr's
- Introduce a new type of instruction format
  - I-type for data transfer instructions (previous format was R-type for **r**egister)
- Example: `lw $t0, 24($s2)`

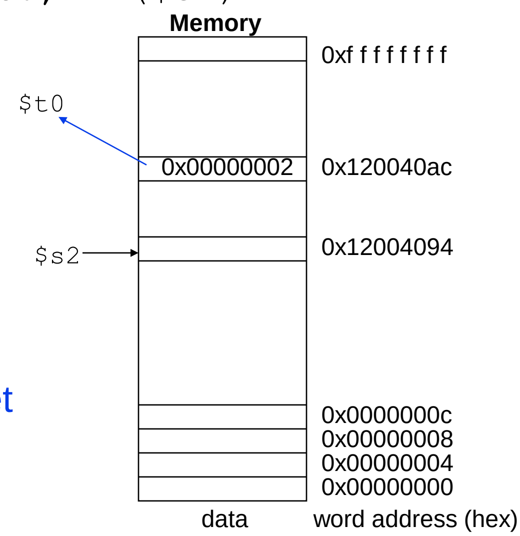


Memory Address Location

- Example: `lw $t0, 24($s2)`

`24`$_{10}$ `+ $s2 =`

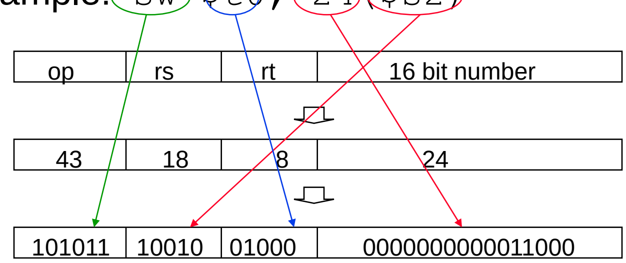
. . . 1001 0100
\+ . . . 0001 1000
. . . 1010 1100 =
0x120040ac



Note that the offset can be positive or negative

Machine Language - Store Instruction

- Example: `sw $t0, 24($s2)`



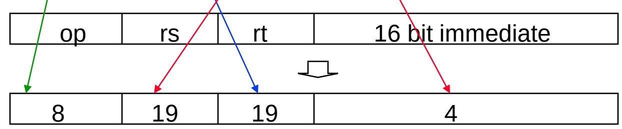
- A 16-bit offset means access is limited to memory locations within a range of $+2^{13}-1$ to -2^{13} (~8,192) words ($+2^{15}-1$ to -2^{15} (~32,768) bytes) of the address in the base register `$s2`
  - 2's complement (1 sign bit + 15 magnitude bits)

Machine Language – Immediate Instructions

- What instruction format is used for the `addi` ?

```
addi $s3, $s3, 4    #$s3 = $s3 + 4
```

- Machine format:



I format

- The constant is kept inside the instruction itself!
  - So must use the I format – Immediate format
  - Limits immediate values to the range $+2^{15}–1$ to -2^{15}

Instruction Format Encoding

- Can reduce the complexity with multiple formats by keeping them as similar as possible
  - First three fields are the same in R-type and I-type
- Each format has a distinct set of values in the op field

Instr	Frmt	op	rs	rt	rd	shamt	funct	address
`add`	R	0	reg	reg	reg	0	32_{ten}	NA
`sub`	R	0	reg	reg	reg	0	34_{ten}	NA
`addi`	I	8_{ten}	reg	reg	NA	NA	NA	constant
`lw`	I	35_{ten}	reg	reg	NA	NA	NA	address
`sw`	I	43_{ten}	reg	reg	NA	NA	NA	address

Assembling Code

- Remember the assembler code we compiled last lecture for the C statement

```
              A[8] = A[2] - b
lw    $t0, 8($s3)      #load A[2] into $t0
sub   $t0, $t0, $s2    #subtract b from A[2]
sw    $t0, 32($s3)     #store result in A[8]
```

- Assemble the MIPS object code for these three instructions (decimal is fine)

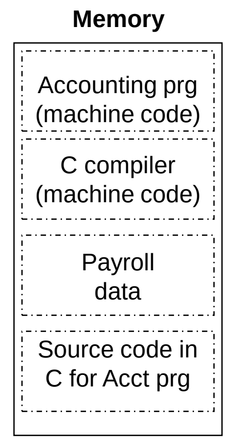
lw	35	19	8	8		
sub	0	8	18	8	0	34
sw	43	19	8	32		

Review: MIPS Instructions, so far

Category	Instr	Op Code	Example	Meaning
Arithmetic (R format)	add	0 & 32	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	0 & 34	sub $s1, $s2, $s3	$s1 = $s2 - $s3
Arithmetic (I format)	add immediate	8	addi $s1, $s2, 4	$s1 = $s2 + 4
Data transfer (I format)	load word	35	lw $s1, 100($s2)	$s1 = Memory($s2+100)
	store word	43	sw $s1, 100($s2)	Memory($s2+100) = $s1

Two Key Principles of Machine Design

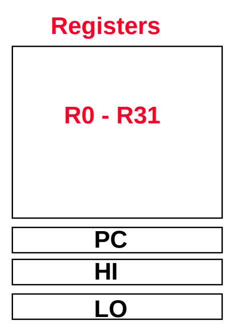
1. Instructions are represented as numbers
2. Programs are stored in memory to be read or written, just like numbers



- Stored-program concept
  - Programs can be shipped as files of binary numbers – binary compatibility
  - Computers can inherit ready-made software provided they are compatible with an existing ISA – leads industry to align around a small number of ISAs

Review: MIPS R3000 ISA

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special



- 3 Instruction Formats: all 32 bits wide

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
OP	rs	rt	rd	shamt	funct	R format
OP	rs	rt	16 bit number			I format
OP	26 bit jump target					