EE 3613: Computer Organization Chapter 2: Instruction Set Architecture – 1/3

Avinash Karanth
Department of Electrical Engineering & Computer Science
Ohio University, Athens, Ohio 45701
E-mail: karanth@ohio.edu
Website:
http://oucsace.cs.ohiou.edu/~avinashk/classes/ee461a/ee461a.htm

1

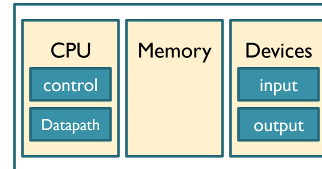
Course Administration

- Lecture Notes 1 (Introduction), 2 (Performance) and 3 (Instruction Set Architecture) posted
- Homework #1 posted, due next Sept 9, 2020 in-class

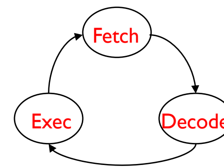
2

(Von Neumann) Processor Organization

- **Control** needs to
  - Input instructions from memory
  - Issue signals to control the information flow between datapath components and to control what operations they perform
  - Control instruction sequencing

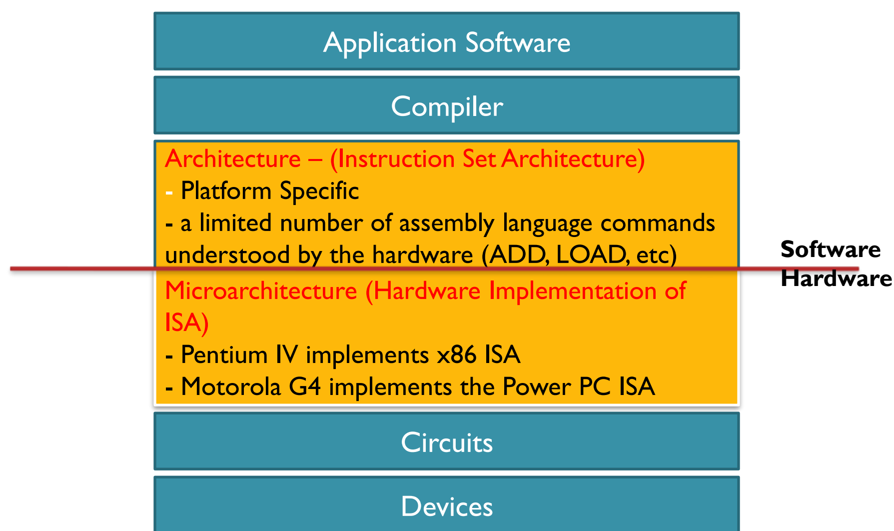


- **Datapath** needs to have the
  - Components – the functional units and storage needed to execute instructions
  - Interconnects – components connected so that instructions can be routed, and data loaded from and stored into memory



3

Where do ISA fit in a computing system?



4

Instruction Set Design (1/2)

- What instructions should be included?
  - Add, Multiply, Divide, Srqt [functions]
  - Branch [flow control]
  - Load/store [storage management]
- What storage locations?
  - How many registers?
  - How much memory?
- How should instructions be formatted?
  - 0, 1, 2 or more operands
  - Immediate operands

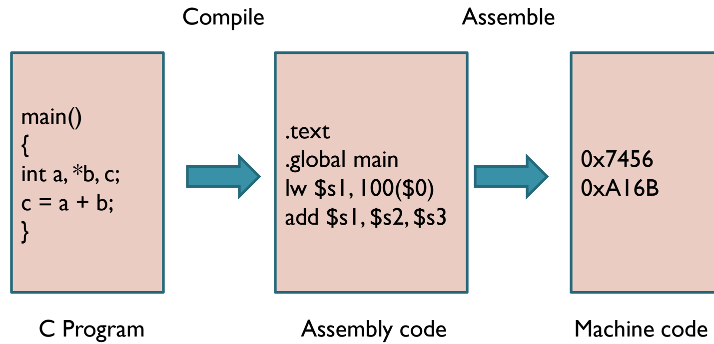
5

Instruction Set Design (2/2)

- How to encode instructions?
  - **RISC** (Reduced Instruction Set Computer)
    - All instructions are the same length (Eg: MIPS, PowerPC, Sun UltraSparc, XAP Processor, ARM processor)
  - **CISC** (Complex Instruction Set Computer)
    - Instructions can vary in size (Eg. VAX, Intel x86)
- What instructions can access memory?
  - For MIPS, only load/store can access memory (load-store architecture)
  - We will be working with MIPS architecture set

6

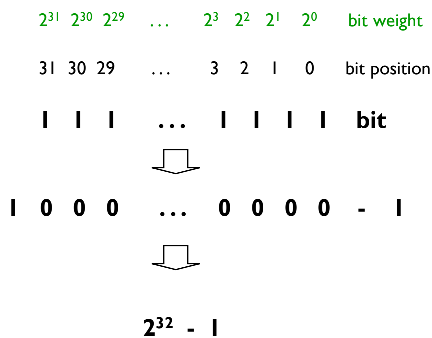
Software Program to Machine Code



7

Unsigned Binary Representation

Hex	Binary	Decimal
0x00000000	0…0000	0
0x00000001	0…0001	1
0x00000002	0…0010	2
0x00000003	0…0011	3
0x00000004	0…0100	4
0x00000005	0…0101	5
0x00000006	0…0110	6
0x00000007	0…0111	7
0x00000008	0…1000	8
0x00000009	0…1001	9
	…	
0xFFFFFFFC	1…1100	2^{32} - 4
0xFFFFFFFD	1…1101	2^{32} - 3
0xFFFFFFFE	1…1110	2^{32} - 2
0xFFFFFFFF	1…1111	2^{32} - 1



8

ASCII: Beyond Numbers

- American Std Code for Info Interchange (ASCII): 8-bit bytes representing characters

ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char
0	Null	32	space	48	0	64	@	96	`	112	p
1		33	!	49	1	65	A	97	a	113	q
2		34	“	50	2	66	B	98	b	114	r
3		35	#	51	3	67	C	99	c	115	s
4	EOT	36	$	52	4	68	D	100	d	116	t
5		37	%	53	5	69	E	101	e	117	u
6	ACK	38	&	54	6	70	F	102	f	118	v
7		39	‘	55	7	71	G	103	g	119	w
8	bksp	40	(	56	8	72	H	104	h	120	x
9	tab	41	)	57	9	73	I	105	i	121	y
10	LF	42	*	58	:	74	J	106	j	122	z
11		43	+	59	;	75	K	107	k	123	{
12	FF	44	,	60	<	76	L	108	l	124	|
15		47	/	63	?	79	O	111	o	127	DEL

9

Architecture Specification

- Data Types
  - Bit, byte, signed/unsigned, logical, floating point, character
- Operations
  - Data movement, arithmetic, shift/rotate, conversion, input/output, control, system calls
- # of operands
  - 3, 2, 1, or 0 operands
- Registers
  - Integer, floating point, control
- Storage for Operands
  - Registers, memory locations, stack locations, fixed registers, fixed location

10

Assembly Code

- a.k.a. Register-transfer-language (RTL)
- Fields
  - **Opcode** – what instruction to perform
  - **Source** – input operand specifiers
  - **Destination** – output operand specifiers
    - What data to perform operation on

Opcode	Destination	Source 1	Source 2
ADD	**$t0**	**$s1**	**$s2**

- Translation – value of $s1 added to $s2 put in $t0

11

MIPS Arithmetic Instructions

- MIPS assembly language arithmetic statement

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

- Each arithmetic statement performs only one operation
- Each arithmetic instruction fits in 32 bits and specifies exactly three operands
- Those operands are all contained in the datapath's register file ($t0, $s1, $s2) – indicated by $
- Operand order is fixed (destination first)

Design Principle 1: Simplicity favors regularity

12

Machine Language – Add Instruction

- Instructions like registers and words of data are all 32 bits long
- Arithmetic instruction Format (R Format): `add $t0, $s1, $s2`

op: 6 bits – opcode that specifies the instruction
rs: 5 bits – register file address of the first source operand
rt: 5 bits – register file address of the second operand
rd: 5 bits – register file address of the result's destination
shamt 5 bits – shift amount (for shift instructions)
funct 6 bits – function code augmenting the opcode

13

Assembly Code Example

- What are the contents of the registers after executing the given assembly code?

Program:
```
add $s1, $s2, $s3
multi $s3, $s3, 3
sub $s2, $s3, $s2
```

Initial Register File

$s1	25
$s2	-4
$s3	57

`add $s1, $s2, $s3`

$s1	
$s2	
$s3	

`multi $s3, $s3, 3`

$s1	
$s2	
$s3	

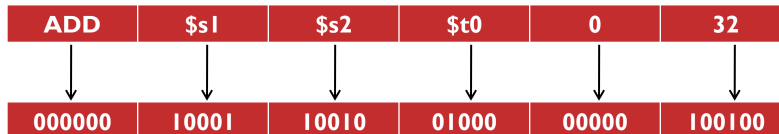
`sub $s2, $s3, $s2`

$s1	
$s2	
$s3	

14

Assembly Instruction Encoding

- Since EDSAC (1949), almost all computers stored program instructions the same way as they store data
- Each instruction is encoded as a number



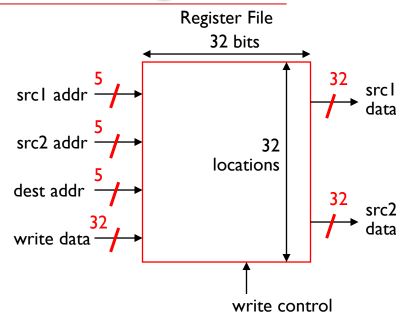
- m bits can encode 2^m different values
- n values can be encoded in $[\log_2(n)]$ bits
- For above example, we can have __ opcodes and __ registers
- And for the above example, the code is _________ in hexadecimals

Storage Architecture

- Registers
  - Fast and small (and useful)
- Immediate values
  - Specifying constants in instructions
- Memory
  - Big and complex (and useful)

MIPS Register Storage

- Holds thirty-two 32-bit registers
  - 2 read ports, 1 write port



- Registers are
  - Faster than main memory
    - But register files with more locations are slower
    - Read/write port increase impacts speed quadratically

Design Principle 2: Smaller is faster

  - Easier for a compiler to use
    - Eg: (A – B) – (C x D) – (E x F) can do multiplies in any order
  - Can hold variables so that
    - Code density improves (since registers are named with fewer bits than a memory location

17

Immediate Values

- Small constant values placed in instructions
- They are stored in memory only because all instructions are in memory (traditionally, not in MIPS)
- In MIPS, constants are built into the instruction having a single operand
  - Example: ptr++; → `addi $s1, $s1, #4`
  - Useful for branch instructions
    - → target address is often immediate in the instruction

Design Principle 3: Make the common case fast

- Size of the immediate is usually determined by how many bits are left in the instruction format

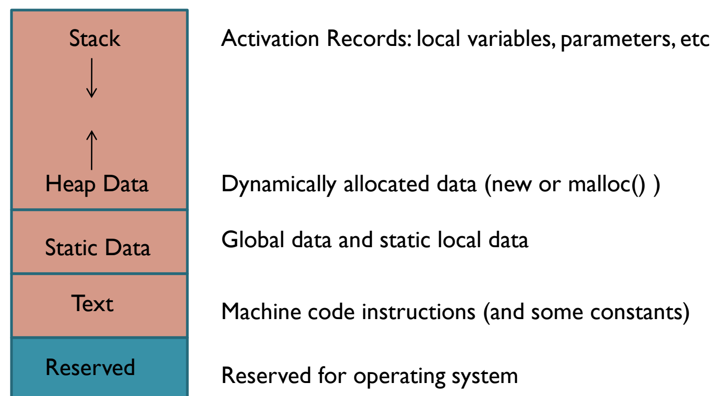
18

Memory Storage

- Large array of storage accessed using memory addresses
  - A machine with a 32 bit addresses can reference memory locations starting from 0 to $2^{32} - 1$ (or 4,294, 967, 295)
  - A machine with a 64 bit address can reference memory locations starting from 0 to $2^{64} - 1$ (or 18,446,744,073, 709, 551, 615)
- Lots of different ways to calculate the addresses

19

Memory Architecture: The MIPS Memory Image



20

MIPS Memory Access Instructions

- MIPS has two basic data transfer instructions to access memory

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

- The data is loaded into (lw) or stored from (sw) a register in the register file – a 5-bit address
- The memory address – a 32-bit address – is formed by adding the contents of the base address register to the offset value
  - A 16-bit field meaning access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of address in the base register
  - Note that offset can be positive or negative

Machine Language – Load Instruction

- Load/Store Instruction Format (I format): `lw $t0, 24($s2)`