Introduction to Computers and Programming

Prof. I. K. Lundqvist

Recitation 2
Sept 18 2003

Some suggestions ... C5, C6, C7

- Binary, Decimal, Hex, ASCII
- Negative numbers / 2's complement
- Floating point numbers
  - Excess
- The Brookshear Machine
  - The Brookshear Instruction Format
  - Brookshear Machine Instructions
- Bit-wise Logical Operations
  - Masking
- The Simple Simulator
- SimpleSim examples
  - Add three numbers together
  - Converting lowercase characters to uppercase

Assembly programs

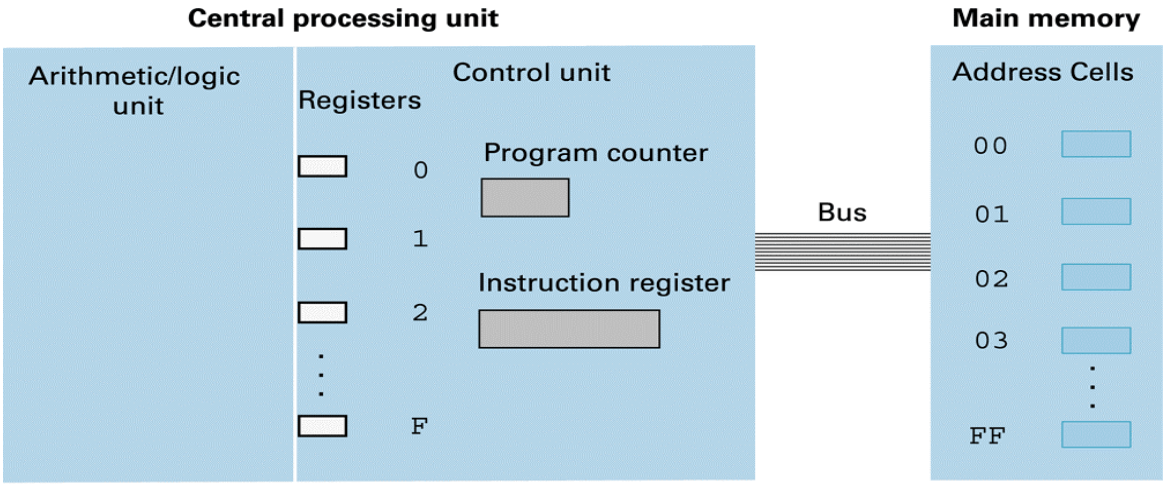
- **Example**: Add three numbers together. The values to added should be placed in memory locations 10h, 11h and 12h. The sum should be placed in memory position 13h.

- **Example**: Write a program that converts lowercase to uppercase ASCII character stored in a memory.

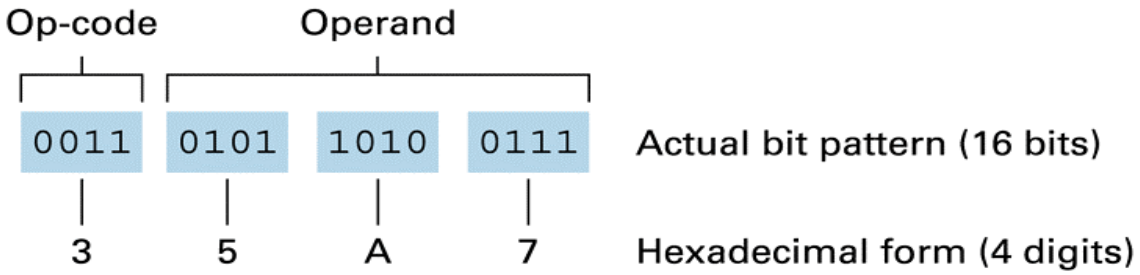
The Brookshear Machine and SimpSim Assembly

- 16 General purpose registers
  - Numbered 0-F (0-15)
- 256 8-bit-size memory cells
  - Numbered 00-FF (0-255)
- 15 simple instructions
  - Encoded using 2 bytes per instruction
  - Hex notation: 4 hex digits per instruction
    - MSB hex for the opcode
    - The other three hex digits for the operands

The Brookshear Architecture



Brookshear Instruction Format



Brookshear Machine Instructions

Op-code	Operation
1	LOAD R,[XY] ;contents
2	LOAD R,XY ;value
3	STORE R,[XY]
4	MOVE S,R
5	ADDI R,S,T ;2's complement
6	ADDF R,S,T ;floating point
7	OR R,S,T
8	AND R,S,T
9	XOR R,S,T
A	ROR R,X
B	jmpEQ R=R0,XY jmp XY
C	Halt
D	Load R,[S]
E	Store R,[S]
F	JumpLE R<=R0,X

BACK

Bit-wise Logical Operations

```
  1001001          11001001
and 1100111      or 01100111
  1000001          11001111
```

```
  100101101
xor 110010011
  010111110
```

Masking: to test the individual pattern of bits in a string of bits

Masking: Reading

- **Example**: Determine if a number is odd or even.

```
        Even Number                 Odd Number
        100101010                   1010100101   <-- number
AND     000000001           AND     0000000001   <-- bit-mask
        000000000                   0000000001   <-- result
```

By examining the result of the masking operation, you can determine the number as odd or even: If the result is 0, it is even

- **Exercise**: Determine if a byte (8 bits in memory) in 2's complement represents a positive or a negative number. What **bit-mask** and what logic operation would you use?

Masking: Setting

- An **OR** operation can be used to set a bit in a string to the value 1.

- **Example**: Change the MSB in a byte to a **1**.

```
        00100110
OR      10000000
        10100110
```

- **Exercise**: If you want to change the bit stream **10011101** into becoming **11011101**, what bit-mask, and what Boolean operator would you use?

Masking: Clearing/Re-setting

- To change a bit in a bit string to value 0 can be done using an AND operation with a 0 in the position that needs to be set to 0 and a 1 in all other positions.
- **Example**:

```
      10100110
AND   01111111
      00100110
```

Masking: Exercises

1. What mask and operator is needed to convert lower case ASCII character to uppercase character? E.g., ‘u’ should be changed to ‘U’, ‘n’ to ‘N’ and so on…
   Hint: take a look at the ASCII values for characters represented in binary notation [ASCII for ‘a’ = 0110 0001, ASCII for ‘A’ = 0100 0001, etc.]

2. Given two bytes. Create a third byte that combines the first half of the 1st byte (4 bits) with the last half of the 2nd byte (4 bits).
   – For example, given **01101001** and **11100111**, the answer would be **01100111**.

   What sequence of logical operations using bit masks are needed to do this?

BACK

Excess Notation

With N bits representation of the exponent we have excess 2^{N-1}

3 bits => excess 4 ($=2^{3-1}$)
4 bits => excess 8 ($=2^{4-1}$)
8 bits => excess 128 ($=2^{8-1}$)

Bit Pattern	Value Represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	-1
0110	-2
0101	-3
0100	-4
0011	-5
0010	-6
0001	-7
0000	-8

Bit Pattern	Value Represented
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

ASCII

A	0100 0001	a	0110 0001
B	0100 0010	b	0110 0010
C	0100 0011	c	0110 0011
D	0100 0100	d	0110 0100
E	0100 0101	e	0110 0101
F	0100 0110	f	0110 0110
G	0100 0111	g	0110 0111
H	0100 1000	h	0110 1000
I	0100 1001	i	0110 1001
J	0100 1010	j	0110 1010
K	0100 1011	k	0110 1011
L	0100 1100	l	0110 1100
M	0100 1101	m	0110 1101
N	0100 1110	n	0110 1110
O	0100 1111	o	0110 1111
P	0101 0000	p	0111 0000
Q	0101 0001	q	0111 0001
R	0101 0010	r	0111 0010
S	0101 0011	s	0111 0011
T	0101 0100	t	0111 0100
U	0101 0101	u	0111 0101
V	0101 0110	v	0111 0110
W	0101 0111	w	0111 0111
X	0101 1000	x	0111 1000
Y	0101 1001	y	0111 1001
Z	0101 1010	z	0111 1010

BACK