CMP 1203

LECTURE 3

Memory Organization and Addressing

- Imagine memory as a matrix of bits
- Each row implemented by a register, length is = addressable unit size of the machine (word size)
- Each register (memory location) has a unique address usually starting from 0
- Addresses usually represented by unsigned integers
- Usually memory is byte addressable but some machines may have a word size > 1 byte

Memory Organization and addressing

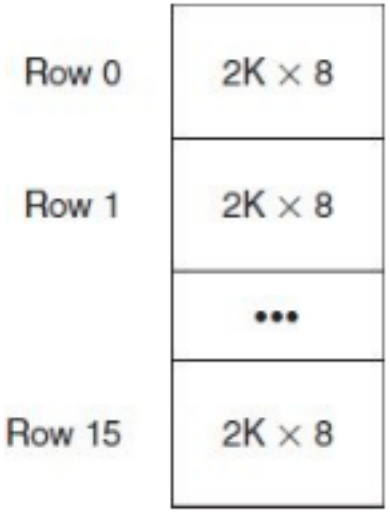
- Memory made from RAM chips
- Notation usually **length X width** e.g. **4M × 8** means memory is 4M long ($2^2 \times 2^{20} = 2^{22}$ items) and each item is 8 bits wide
- **Qn:** How many unique addresses do we need?
- Ans: 2^{22} , counting from 0 to (2^{22} -1) in binary
- **Qn**: So how many bits do we need?
- 22

Memory Organization and addressing

- Now assume this memory is instead word addressable and a word is 16 bits.
- How many bits required per address?
- **Ans**: ($2^{21} \times 16$) ➔ 2^{21} words ➔ 21 bits
- Main memory is usually bigger than one RAM chip so combine multiple chips to get a single memory of the specified size
- E.g. build 32K × 8 memory from 2K × 8 RAM chips

Memory Organization and Addressing

- Each chip addresses 2K (2^{11}) bytes.
- How many bits needed for this memory's address? 15 ($32K = 2^5 \times 2^{10}$)
- But each chip needs only 11 address lines (each chip holds 2^{11} bytes)
- In such cases, need a decoder to decode either the leftmost 4 bits or rightmost 4 bits of the address to know which chip has the required data (**why 4 bits?**)



Memory Organization and Addressing

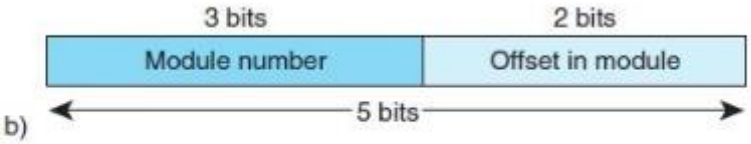
- A single memory module causes sequentialization of access (access only one memory at a time)
- Memory interleaving: splits memory across different modules to allow simultaneous access
- Low- order or high-order interleaving defines whether the low order bits or high order bits of the address are used to select the bank
- Consider a byte-addressable memory made of 8 modules of 4 bytes each , total memory 32 bytes.
- 5 bits used to uniquely i.d each byte, 3 for the module and 2 for the offset within that module

Memory Organization and Addressing

a)

Module 0	Module 1	Module 2	Module 3	Module 4	Module 5	Module 6	Module 7
0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

b)



c)

Module	Decimal Word Address	Binary Address	Address Split per Given Structure	Module Number	Offset in Module
Module 0	0	00000	000 00	0	0
	1	00001	000 01	0	1
	2	00010	000 10	0	2
	3	00011	000 11	0	3
Module 1	4	00100	001 00	1	0
	5	00101	001 01	1	1
	6	00110	001 10	1	2
	7	00111	001 11	1	3

FIGURE 4.6 a) High-Order Memory Interleaving
b) Address Structure
c) First Two Modules

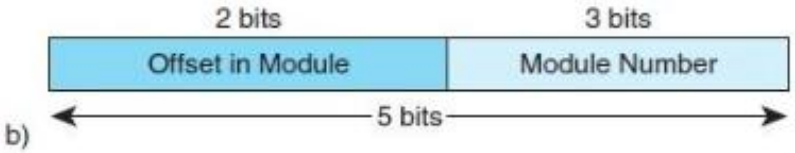
Image source: Lobur and Null ; High – order memory interleaving

Memory Organization and Addressing

a)

Module 0	Module 1	Module 2	Module 3	Module 4	Module 5	Module 6	Module 7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

b)



c)

Module	Decimal Word Address	Binary Address	Address Split per Given Structure	Offset in Module	Module Number
Module 0	0	00000	00 000	0	0
	8	01000	01 000	1	0
	16	10000	10 000	2	0
	24	11000	11 000	3	0
Module 1	1	00001	00 001	0	1
	9	01001	01 001	1	1
	17	10001	10 001	2	1
	25	11001	11 001	3	1

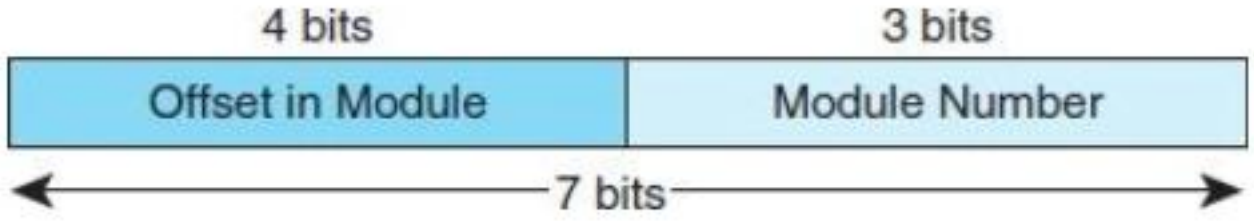
FIGURE 4.7 a) Low-Order Memory Interleaving
b) Address Structure
c) First Two Modules

Image source: Lobur and Null ; Low – order memory interleaving

Question

Consider an 8-way low order interleaved memory whose storage capacity is 128 words.

1. How many bits are needed for each address?
2. How many words does each module store?
3. Draw address structure.



Interrupts

- Events that alter normal flow of program execution in a computer
- Can be triggered due to:

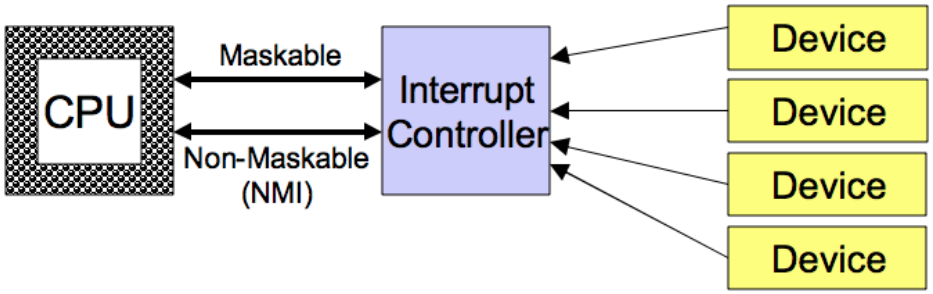
1. I/O requests
2. Arithmetic errors e.g. division by 0
3. Hardware malfunctions
4. User defined break points etc.

- Initiated by a user or system
- Maskable (disabled or ignored) or unmaskable (must be acknowledged)
- Synchronous (occurs at same place each time a program is executed) or asynchronous (occur unexpectedly)

Interrupts

- I/O example
- Devices need a way to inform CPU that they need to use it. CPU can't predict when.

1. Polling- routinely query each device – wastes CPU time but can be good if need urgent response
2. Give each device a wire that it uses to signal CPU when it needs to use it. Processor executes a routine – "interrupt handler" to deal with it



DATA REPRESENTATION

Representing Signed Integers: Signed Magnitude

- Signed integers (set of positive and negative integers)
- First bit represents the sign 1 (-) and 0 (+)
- So N bits can represent

$$-(2^{(N-1)}-1) \textit{ to } (2^{(N-1)}-1)$$

Arithmetic Rules:

- If same signs, add the magnitudes and then use same sign for result
- If different signs, determine the operand with the larger magnitude and sign is the same as the largest operand

Representing Signed Integers: Signed Magnitude

EXAMPLE 2.10 Add 01001111_2 to 00100011_2 using signed-magnitude arithmetic.

$$\begin{array}{rcl} & 1\ 1\ 1\ 1 & \Leftarrow \text{carries} \\ 0 & 1\ 0\ 0\ 1\ 1\ 1\ 1 & (79) \\ 0 + & 0\ 1\ 0\ 0\ 0\ 1\ 1 & +\ (35) \\ \hline 0 & 1\ 1\ 1\ 0\ 0\ 1\ 0 & (114) \end{array}$$

- Example 2: Add 01001111_2 to 01100011_2
- If there's a carry in the 7^{th} bit, it is discarded and we say an overflow has occurred
- This gives an incorrect value
- Programmers need to anticipate different overflow cases in program
- This method has 2 representations for 0!

Representing Signed Integers: Signed Magnitude

EXAMPLE 2.10 Add 01001111_2 to 00100011_2 using signed-magnitude arithmetic.

```
          1 1 1 1   ⇐ carries
0   1 0 0 1 1 1 1      (79)
0 + 0 1 0 0 0 1 1    + (35)
-------------------  -------
0   1 1 1 0 0 1 0      (114)
```

- Example 2: Add 01001111_2 to 01100011_2

Representing Signed Integers: Complement Systems

- In decimal: A–B is done by subtracting B from all 9's, add difference to A, and then add the carry.
- This is called finding 9's complement or finding the **diminished radix complement**
- E.g. To find 873–218; 999–218=781; 873+781 = 1654: Drop leading 1 and add it back to last digit to give 655
- Extended to binary to simplify computer arithmetic

One's Complement

- One's complement is got by subtracting from 1
- E.g. one's complement of **0101** is 1111–0101=**1010**
- For binary : **simply flip all bits in number to get 1's complement**
- Since we don't want to use an extra bit to represent the sign, we simply complement only negative numbers

Example: Express 23_{10} and -9_{10} in 8-bit binary, if the computer is using 1's complement representation

Addition and Subtraction

EXAMPLE 2.17 Add 01001111_2 to 00100011_2 using one's complement addition.

```
                1 1 1 1     ⇐ carries
  0 1 0 0 1 1 1 1   (79)
+ 0 0 1 0 0 0 1 1 + (35)
  0 1 1 1 0 0 1 0   (114)
```

EXAMPLE 2.18 Add 23_{10} to -9_{10} using one's complement arithmetic.

```
  1 ← 1 1 1   1 1       ⇐ carries
      0 0 0 1 0 1 1 1   (23)
    + 1 1 1 1 0 1 1 0 + (−9)
      0 0 0 0 1 1 0 1
                  + 1
      0 0 0 0 1 1 1 0   14₁₀
```

The last carry is added to the sum.

Two's Complement

- 1's complement still has two representations for 0 (00000000 and 11111111)!
- Radix complement for a number N in base r having d digits is defined as $r^d - N$ for N not equal to 0, and 0 for N = 0.
- E.g. 2's complement of 0011_2 is $2^4 - 0011_2 = 10000_2 - 0011_2 = 1101_2$
- **Observe that 2's complement is simply 1's complement and add 1**

Examples

- Simplifies add and subtract because there are no carry-ends to worry about
- Simply discard carries involving higher order bits

EXAMPLE 2.20 Express 23_{10}, -23_{10}, and -9_{10} in 8-bit binary, assuming a computer is using two's complement representation.

$$23_{10} = +(00010111_2) = 00010111_2$$
$$-23_{10} = -(00010111_2) = 11101000_2 + 1 = 11101001_2$$
$$-9_{10} = -(00001001_2) = 11110110_2 + 1 = 11110111_2$$

EXAMPLE 2.21 Add 01001111_2 to 00100011_2 using two's complement addition.

			1	1	1	1	⇐ carries	
	0	1	0	0	1	1	1	1	(79)
	0	0	1	0	0	0	1	1	+ (35)
+	0	1	1	1	0	0	10		(114)

Examples

- Add a) 23_{10} and -9_{10} and b) -23_{10} and -9_{10}

EXAMPLE 2.23 Find the sum of 23_{10} and -9_{10} in binary using two's complement arithmetic.

	$1\leftarrow$	1	1	1		1	1	1	$\Leftarrow$ carries
Discard		0	0	0	1	0	1	1	(23)
carry.	+	1	1	1	1	0	1	1	+ (−9)
		0	0	0	0	1	1	1	14_{10}

In two's complement, the addition of two negative numbers produces a negative number, as we might expect.

EXAMPLE 2.24 Find the sum of 11101001_2 (–23) and 11110111_2 (–9) using two's complement addition.

	$1\leftarrow$	1	1	1	1	1	1	1	$\Leftarrow$ carries
Discard		1	1	1	0	1	0	0	1 (−23)
carry.	+	1	1	1	1	0	1	1	1 + (−9)
		1	1	1	0	0	0	0	0 (−32)

Overflow Detection

- Simple rule:
- If carry into the sign bit is equal to the carry out of the bit, no overflow has occurred. IF they are not equal, overflow (thus errors) have occurred

EXAMPLE 2.25 Find the sum of 126_{10} and 8_{10} in binary using two's complement arithmetic.

	$0\leftarrow$	1	1	1	1				$\Leftarrow$ carries
Discard last		0	1	1	1	1	1	1	0 (126)
carry.	+	0	0	0	0	1	0	0	0 +(8)
		1	0	0	0	0	1	1	0 (−122???)

A one is carried into the leftmost bit, but a zero is carried out. Because these carries are not equal, an overflow has occurred. (We can easily see that two positive numbers are being added but the result is negative.) We return to this topic in Section 2.4.6.

Integer Multiplication and Division

- Simplest multiplication method similar to decimal multiplication on paper
- $0 \times A = 0$ and $1 \times A = 1$
- Need 3 storage areas: multiplicand, multiplier and product
- Start with LSB, for each digit in multiplier, shift the multiplicand one bit to the left. If multiplier is a 1, add shifted value to the running sum of partial products
- In actual computer, more sophisticated methods are used to optimize process e.g. carrying out some parts in parallel

Integer Multiplication and Division

Example Consider the multiplication of the two unsigned numbers 14 and 10. The process is shown below using the binary representation of the two numbers.

```
        1110 (14) Multiplicand( M)
        1010 (10) Multiplier(Q)

        ------
         0000     (Partial Product)
        1110      (Partial Product)
       0000       (Partial Product)
      1110        (Partial Product)
      ======
      10001100 (140) Final Product(P)
```

Homework:

1. Review simple binary division
2. Advantages and disadvantages of signed-magnitude Vs 2's complement representation of signed number.

Integer Multiplication: Add Shift Method

- Series of (n) add and shift operations
- If bit of multiplier is 0, only shift operation and if 1, both add and shift

Example

- Consider multiplication of 2 unsigned numbers 11 and 13.
- Assume a is a 4-bit register and initialized to 0's
- C is the carry bit from the MSB position
- Process repeated n = 4 times (number of bits in the multiplier Q)
- If the bit of the multiplier is 1, the A ← A + M and the concatenation of AQ is shifted 1 bit to the right.
- If the bit is 0, then only shift AQ

Add and Shift

M	*C*	*A*	*Q*		
1011	0	0000	1101	Initial values	
1011	0	1011	1101	Add	First cycle
1011	0	0101	1110	Shift	
1011	0	0010	1111	Shift	Second cycle
1011	0	1101	1111	Add	Third cycle
1011	0	0110	1111	Shift	
1011	1	0001	1111	Add	Fourth cycle
1011	0	1000	1111	Shift	

Lecture Handout

- Read Chapter 1 and 4 of Null and Lobur