Computer Architecture and Microprocessor

김 동 한
donghani@khu.ac.kr

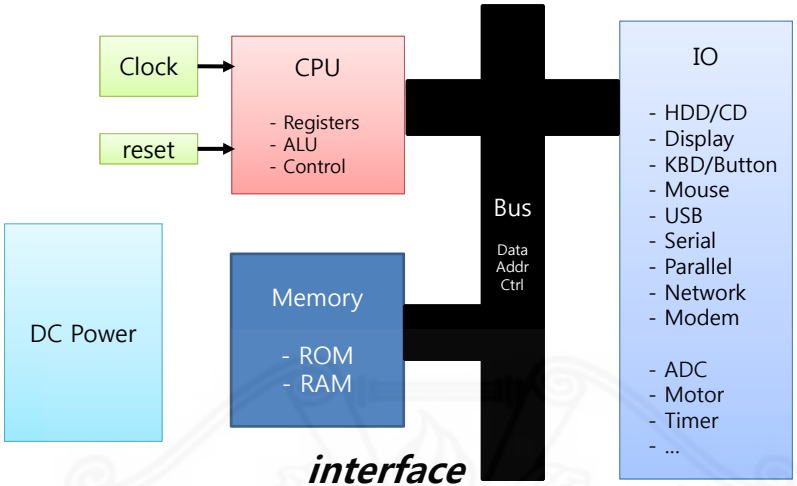
Computer Category

- Desktop (PC, Workstation)
  - Optimized for single user (personal computer)
  - Applications → memory size, disk space, footprint
  - Design goal: max performance for a given power/price
- Servers
  - Old story: minicomputers, mainframes, supercomputers
  - Multiple users/applications
  - Reliability(Dependability), availability (percent of up-time) critical, expandability
  - Throughput is the main performance measure
  - Design goal: carry large workloads: either single complex applications (e.g., scientific applications) or many small jobs (e.g. Web server)
- Notebook computers
  - "Mobile Desktop"
  - Battery life, low power, compact
- **Embedded computers**
  - Application specific, tuned, price and power sensitive, volume
  - Often with **real-time** computing constraints
  - System on Chip (SoC) – on die CPU core + custom hardware
  - Examples: mobile phone, PDA, playstation, set-top box, digital appliances
- Ultra-Mobile Personal Computers (UMPC)
  - Power envelope (no ventilation)
  - User interface customization

Function

- All computer functions are comprised of four basic operations:
  - Data processing
  - Data storage
  - Data movement
  - Control

Computer


Clock
reset
CPU
- Registers
- ALU
- Control
Bus
Data
Addr
Ctrl
IO
- HDD/CD
- Display
- KBD/Button
- Mouse
- USB
- Serial
- Parallel
- Network
- Modem
- ADC
- Motor
- Timer
- ...
DC Power
Memory
- ROM
- RAM
interface

Components of a Computer

- CPU : Registers + ALU(arithmetic logic unit) + Control Unit
- Bus
  - Data and address, control
- Memory
  - Internal(on-chip) and external
  - ROM(PROM, EPROM) and RAM
  - DRAM(SDRAM, DDR RAM) and SRAM
  - Cache, Main memory, Virtual memory
- IO
  - Serial, Parallel, USB
  - Graphic display (Frame memory + DAC(R,G,B) + Controller)
  - Storage (HDD, CD)
  - Key board (button), Mouse, …
  - ADC, DAC
  - …
- DC Power
  - 5 v, 3.3 v
  - 9 v, 12 v, 18 v

Computer Architecture

- Architecture:
  - The collection of processor's (or a system's) features as they are seen by the "user"

- Registers data width (8/16/32/64)
- Instruction set
- Addressing modes and methods (Segmentation, Paging, etc...)
- Memory controllers and hierarchies
- System interconnects such as computer buses
- Multi-processing

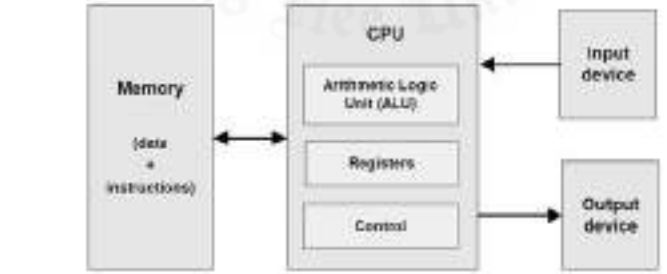
Program Concept

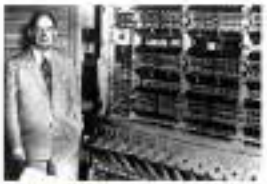
- Hardwired systems are inflexible
- General purpose hardware can do different tasks, given correct control signals
- Instead of re-wiring, supply a new set of control signals

What is a program?

- A sequence of steps
- For each step, an arithmetic or logical operation is done
- For each operation, a different set of control signals is needed

Von Neumann machine (1940)





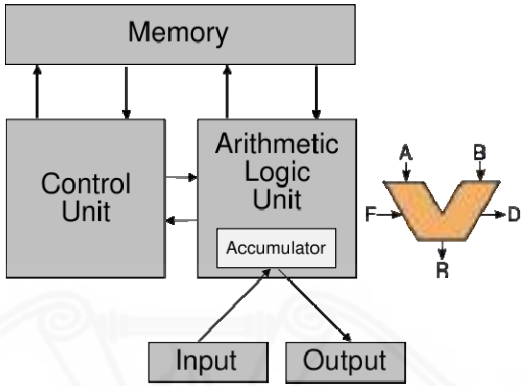
John Von Neumann (and others) ... made it possible

Stored
program
concept!

Andy Grove (and others) ... made it small (and fast).

Von Neumann Architecture

a design model for a **stored-program** digital computer that uses a **processing unit** and a **single separate storage** structure to hold both instructions and data



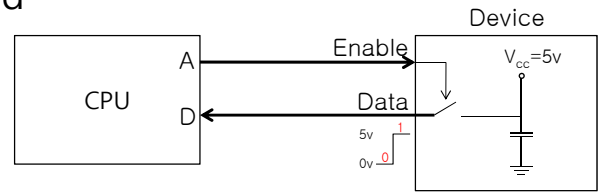
cf) Harvard architecture : PM+DM

CPU (central processing unit)

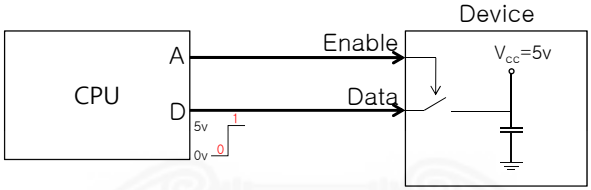
- CPU must
  - Fetch instructions
  - Interpret instructions
  - Fetch data
  - Process data
  - Write data

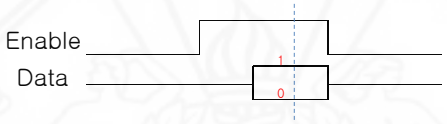
Access (Address and Data)

- Read

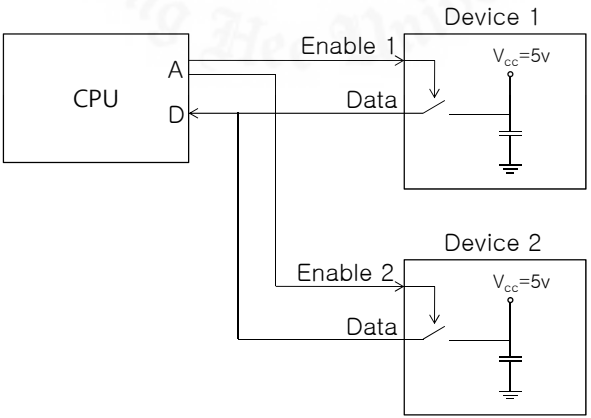


- Write





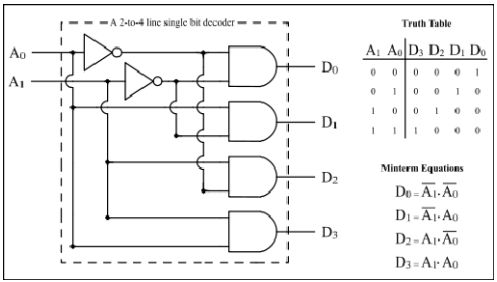
Addressing



100 devices → 100 enable lines
1GB memory → >1,000,000,000 enable lines!

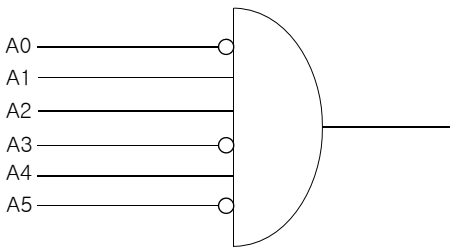
"Address decoding" : 32 address lines → 4G enable lines

Address Decoder (2 to 4)

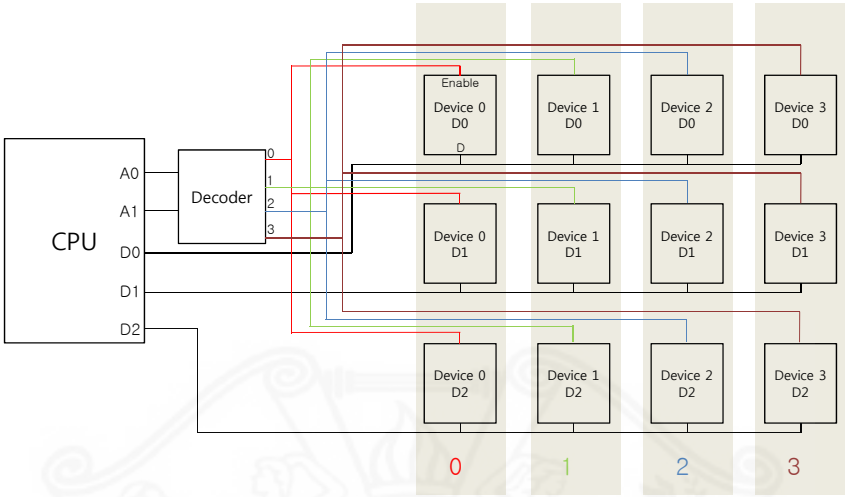


Decoding Example

- Address space : 6 bits (0x00 ~ 0x3F)
- Given address : 0x16 → 0b 01 0110

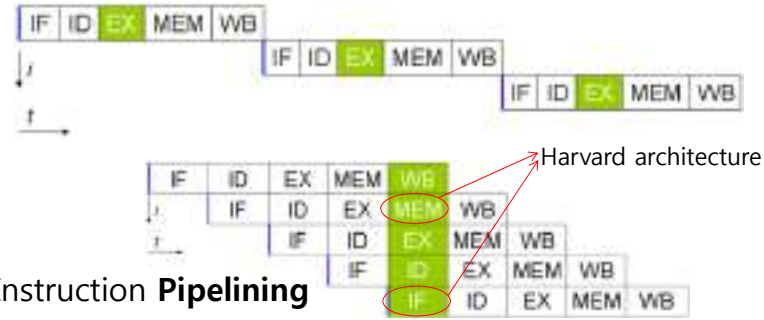


Bus Interface
3-bit CPU



Instruction Subcycles and Pipelining

1. Fetch instruction from memory
2. Read registers while decoding the instruction
3. Execute the operation or calculate a memory address
4. Access an operand in data memory
5. Write the result into a register



Instruction **Pipelining**

→"One instruction / cycle"
"Single clock cycle execution"

CPU

- Functional Units + Registers + Control Unit

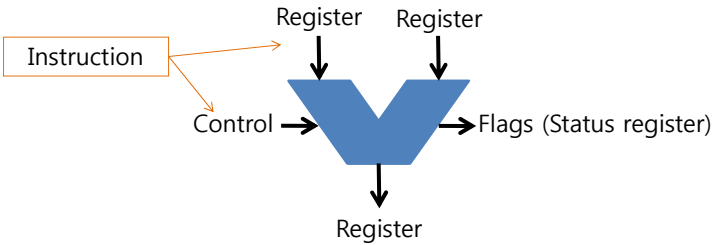
- Functional Units
  - ALU = Arithmetic and Logic Unit
  - Could have many functional units (some special-purpose, e.g., multiply,…)
- Registers
  - Small, temporary storage
  - Operands and results of functional units
- Control Unit
  - Orchestrates execution of the program
  - Program Counter (PC) : contains the address of the next instruction to execute
  - Reads an instruction from memory (at PC)
  - Interprets the instruction
  - Generates signals that tell the other components what to do
  - Instruction may take many machine cycles to complete

- Word Size
  - Number of bits normally processed by ALU in one instruction
  - Also width of registers

ALU (arithmetic logic unit)

- Does the calculations
  - Integer arithmetic operations (addition, subtraction, and sometimes multiplication and division, though this is more expensive)
  - Bitwise logic operations (NOT, AND, OR, XOR)
  - Bit-shifting operations (shifting or rotating a word by a specified number of bits to the left or right, with or without sign extension)

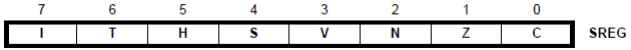
cf) FPU (floating point unit)



Registers

- CPU must have some working space (temporary storage)
- Number and function vary between processor designs

- General purpose registers (accumulator)
  - 32x8 registers (R0~R31) in AVR
  - 6x32 registers in Pentium
  - 14x64 registers in i7
- Program counter (instruction pointer)
- Status (flag) register
  - Carry, Zero, Negative, Overflow, Sign, Interrupt,…
  - For conditional branch



- Stack pointer

- HW control registers

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see *CONTROL register on page 21*. In Handler mode, the processor always uses the main stack. The options for processor operations are:

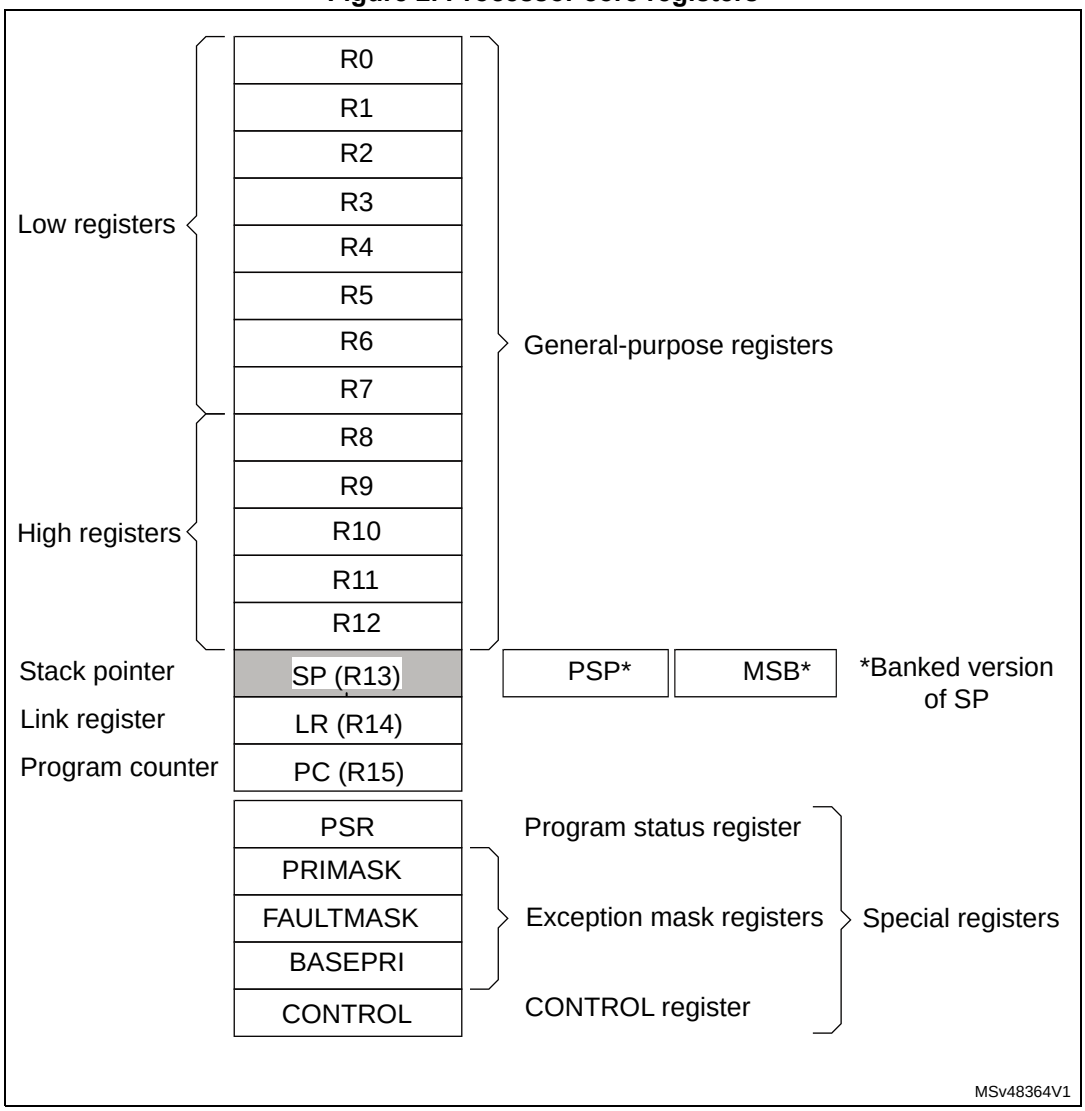
Table 1. Summary of processor mode, execution privilege level, and stack use options

Processor mode	Used to execute	Privilege level for software execution	Stack used
Thread	Applications	Privileged or unprivileged (1)	Main stack or process stack (1)
Handler	Exception handlers	Always privileged	Main stack

1. See *CONTROL register on page 21*.

2.1.3 Core registers

Figure 2. Processor core registers



Instructions

- Fundamental unit of work
- Constituents
  - Opcode: operation to be performed
  - Operands: data/locations to be used for operation
  - e.g.) ADD R0, 3
- Encoded as a sequence of bits (just like data!)
  - Sometimes have a fixed length (e.g., 16 or 32 bits)
  - Control unit interprets instruction
    Generates control signals to carry out operation
  - Atomic: operation is either executed completely, or not at all
- Instruction Set Architecture (ISA)
  - Computer's instructions, their formats, their behaviors

Instruction Set

- Arithmetic instructions such as *add* and *subtract*
- Logic instructions such as *and*, *or*, *xor*, and *not*
- Data instructions such as *move*, *input*, *output*, *load*, and *store*
- Control flow instructions such as *goto*, *if*, *call*, and *return*

CISC and RISC

- CISC
  - Each instruction can execute several low-level operations (microcode).
  - In the early days of the computer industry, programming was done in assembly language or machine code, which encouraged powerful and easy to use instructions.
  - An important force encouraging complexity was very limited main memories.
  - CPUs also had relatively few registers. (Pentium: 8)
  - e.g.) x86, 8051
- RISC
  - load-store architecture
  - One instruction per cycle
  - Large number of general purpose registers (AVR: 32)
  - Limited and simple instruction set (Emphasis on optimizing the instruction pipeline)
  - Register to register operations
  - Few, simple addressing modes
  - Few, simple instruction formats
  - Hardwired design (no microcode)
  - Fixed instruction format
  - More compile time/effort
  - e.g.) AVR, ARM, SPARC(Sun)
- The terms RISC and CISC have become less meaningful with the continued evolution of both CISC and RISC designs and implementations.

Numbers

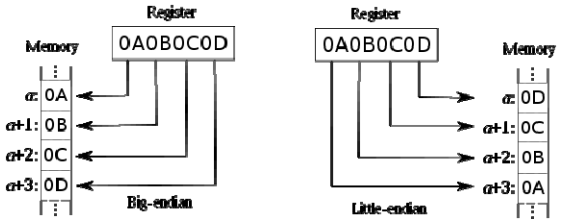
- Only binary numbers (0,1)
- No minus sign

- Integer
  - Fixed point
  - Bits (8/16/32) → byte/short word/long word
  - signed/unsigned → negative number?

- Floating point (real)

Endianness

- Byte ordering in memory → CPU
- MSB and LSB
- Big-endian : big end first
- Little-endian : little end first



- Byte-addressable CPU
- File

- e.g.) x86(PC) : little-endian, PowerPC : big-endian

ARM : Bi-endian

Signed Integer

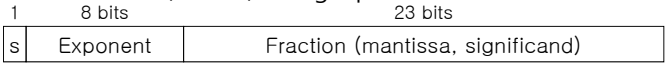
- 1's complement
  - Bitwise NOT (Boolean complement)
  - 1:0000 0001, -1: 1111 1110
  - 1+(-1)=1111 1111?
  - 0: 0000 0000, -0: 1111 1111?
- 2's complement
  - Negating : 1's complement + 1
    - 1 = 0000 0001
    - 1's complement gives 1111 1110
    - Add 1 to LSB 1111 1111 → -1
  - Arithmetic works easily and consistently
    - 1+(-1) = 0000 0000
    - 0-1 = 0000 0000 – 0000 0001 = 1111 1111
  - The MSB is a sign bit.
  - 8 bits : -128~127, 16 bits : -32768~32767
  - 8 bits integer to 16 bits
    - Signed extension
      -1(8) → -1(16) : 1111 1111 → 1111 1111 1111 1111
    - Unsigned extension
      255(8) → 255(16) : 1111 1111 → 0000 0000 1111 1111
  - 0xff (1111 1111) = -1 or 255?

+127 = 0111 1111
...
+3 = 0000 0011
+2 = 0000 0010
+1 = 0000 0001
+0 = 0000 0000
-1 = 1111 1111
-2 = 1111 1110
-3 = 1111 1101
...
-128 = 1000 0000

Floating Point

- Very large and very small numbers
  - Large values: 6.023×10^{23} -- requires 79 bits
  - Small values: 6.626×10^{-34} -- requires >110 bits
- Numbers with fractions
  - Could be done in pure binary
  - $1001.1010 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9.625$
  - Where is the binary point?
- Use equivalent of "scientific notation": $F \times 2^E$

- IEEE 754 standard (32-bits) – single precision



- $N=(-1)^s \times \mathbf{1.}\text{fraction} \times 2^{(\text{exponent}-127)}$ $(1 \le \text{exponent} \le 254)$
- e.g.) 1 01111110 01000000000000000000000
  - Sign is 1: number is negative
  - Exponent field is 01111110 = 126 (decimal)
  - Fraction is 0.010000000000… = $1 \times 2^{-2} = 0.25$ (decimal)
  - Value = $-1.25 \times 2^{(126-127)} = -1.25 \times 2^{-1} = -0.625$
- Double precision (64 bits)
  - 11-bit exponent field, 52-bit fraction field

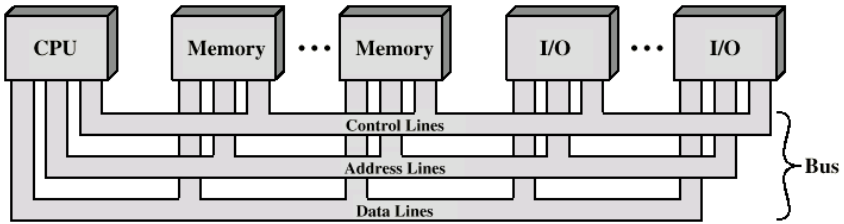
Character

- For text
- A character is also a number.

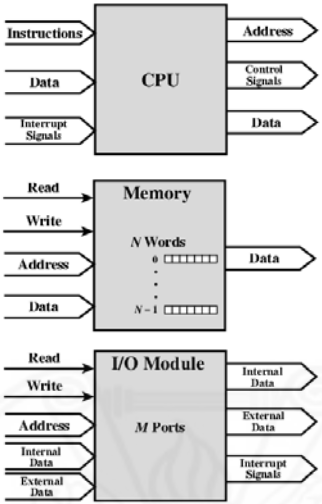
- ASCII code : 1 byte (7 bits; 0~127)
  - 'A'=0x41
  - 'A'+1='B'
- 한글 : 11172자
  - 2 bytes code
  - 완성형 : '가' 0xb0a1
    - KSC5601-1987 : 2350자
  - 조합형
    - KSC5601-1992
- Unicode
  - 0~0x10FFFF
  - 2 bytes - 65535 글자
  - 한글 완성형 11172자 배정
  - 한글 조합형 : Hangul Jamo
  - KSC5700
- String
  - Sequence of characters, terminated with NULL (0)

Bus

- All the units must be connected
- Different type of connection for different type of unit
  - CPU
  - Memory
  - Input/Output
- Single and multiple BUS structures are most common
- Usually broadcast
- Control/Address/Data bus + Power/GND lines
- What do buses look like?
  - Sets of wires
  - Parallel lines on circuit boards
  - Ribbon cables
  - Strip connectors on mother boards



Computer Modules and Bus



Bus

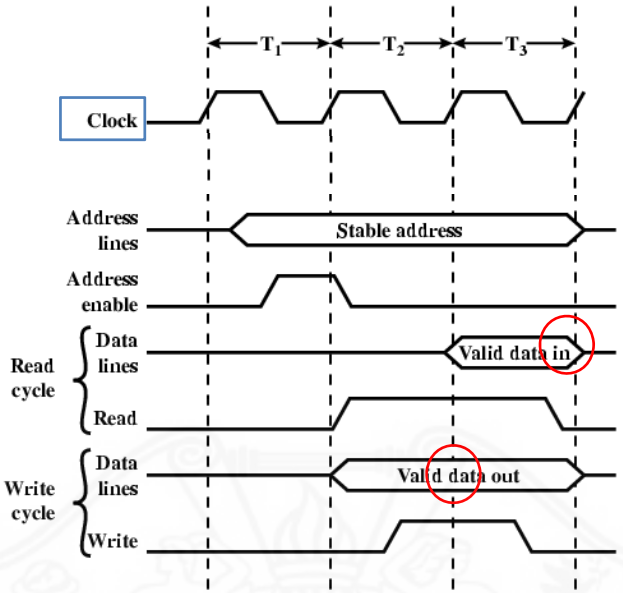
- Data
  - Carries data (including instructions)
  - Width is a key determinant of performance: 8, 16, 32, 64 bits
    - AVR : 8 bits
- Address
  - Identify the source or destination of data
  - CPU needs to read an instruction (data) from a given location in memory.
  - Width determines maximum memory capacity of system.
    - Pentium : 32 bits (4G address space)
      AVR : 16 bits (64K)
- Control
  - Memory read/write signal
    - AVR : /RD, /WR, ALE
  - Interrupt request
    - AVR : INT0:7
  - Clock signals

Bus Arbitration

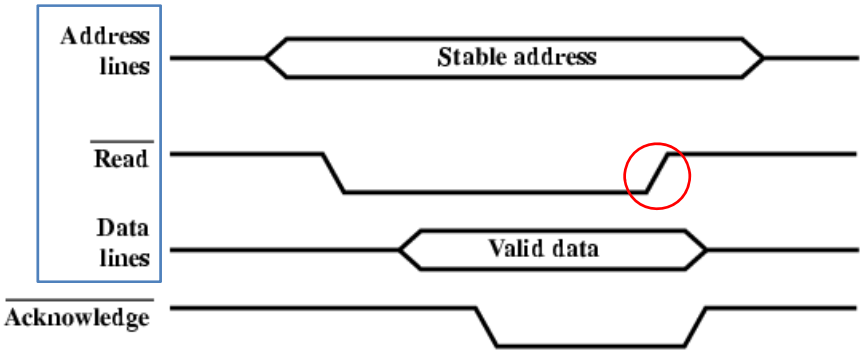
- More than one module controlling the bus
- CPU1, CPU2,..., and DMA controller
- Only one module may control bus at one time
- Arbitration may be
  - Centralized
    - Single hardware device(bus controller, arbiter) controlling bus access
    - May be part of CPU or separate
  - Distributed
    - Each module may claim the bus
    - Control logic on all modules

- cf) PCI (Peripheral Component Interconnection)

Synchronous Timing Diagram



Asynchronous Timing – Read Diagram

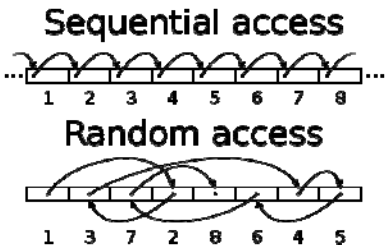


Memory

- Internal(on-chip) and external
- ROM(PROM, EPROM, FLASH) and RAM
- DRAM(SDRAM, DDR RAM) and SRAM
- Cache, Main memory, Virtual memory

RAM (Random Access Memory)

- Misnamed as all semiconductor memory is random access
- Read/Write
- Volatile
- Temporary storage
- Static or dynamic



DRAM vs SRAM

- Both volatile
  - Power needed to preserve data
- Dynamic cell
  - Bits stored as charge in capacitors
  - Charges leak : Needs refresh
  - Simpler to build, smaller (one transistor + one capacitor)
  - More dense
  - Less expensive
  - Larger memory units
- Static
  - Flip-flop (4 to 6 transistors)
  - Faster
  - Cache
- Access time and cost (for 2010)
  - SRAM : 1.5 ns, $6,000 per GB
  - DRAM : 40 ns, $60 per GB
  - Disk : 3 million ns, $0.3 per GB

SDRAM (Synchronous DRAM)

- Access is synchronized with an external clock
- Since SDRAM moves data in time with system clock, CPU knows when data will be ready
- **Burst mode** allows SDRAM to set up stream of data and fire it out in block
- DDR-SDRAM sends data twice per clock cycle (leading & falling edge)

SDRAM Read Timing

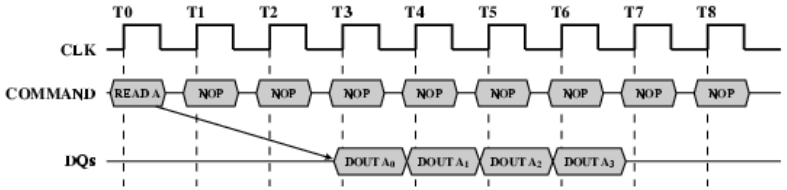


Figure 5.13 SDRAM Read Timing (Burst Length = 4, $\overline{\text{CAS}}$ latency = 2)

Types of ROM

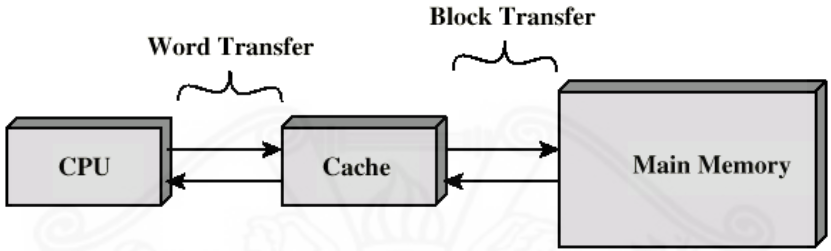
- Written during manufacture
  - Very expensive for small runs
- Programmable (once)
  - PROM
  - Needs special equipment to program
- Read "mostly"
  - Erasable Programmable (EPROM)
    - Erased by UV
  - Electrically Erasable (EEPROM)
    - Takes much longer to write than read
  - Flash memory
    - Erase whole memory electrically

Von Neumann Bottleneck

- The separation between the CPU and memory → the limited throughput (data transfer rate) between the CPU and memory compared to the amount of memory
- In most modern computers, throughput is much smaller than the rate at which the CPU can work. → The CPU is continuously forced to wait for needed data to be transferred to or from memory.
- The performance problem is reduced by a **cache** between the CPU and the main memory.

Cache

- Fast CPU ⇔ Slow main memory
- Small amount of fast memory
- Between normal main memory and CPU
- External or internal
- Data and instruction cache ← Harvard architecture
- Cache miss/hit
- Intel Core i7
  - 32 KB L1 instruction and 32 KB L1 data cache per core
  - 256 KB L2 cache (combined instruction and data) per core
  - 8 MB L3 (combined instruction and data) shared by all cores



IO

- Programmed IO
  - CPU has direct control over I/O
    - Sensing status
    - Read/write commands
    - Transferring data
  - CPU waits for I/O module to complete operation
  - Wastes CPU time

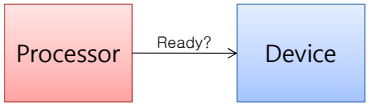
- DMA (direct memory access)
  - Memory/IO accesses are done independently of CPU.
  - IO to IO, IO to memory, memory to memory
  - A DMA controller(HW) is required. (Bus arbitration)
  - Fast block transfer
  - No burden to CPU

Addressing IO

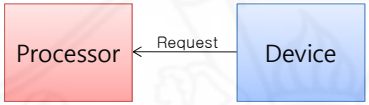
- Memory-mapped IO
  - Memory and I/O share the same address space.
    (IO address space is part of memory address space.)
  - IO reads and writes are done in the same way as memory
    mov ax, [0x200], mov [0x300], ax
  - Decoding : address bus + R/W(memory)
- IO-mapped IO
  - IO has its own address space. (A CPU has extra IO read/write pins)
  - Decoding : address bus + R/W(IO)
  - A CPU has IO instructions (ex: in, out)
    in ax, 0x200, out 0x300, ax
  - Generally slow

Interacting with external devices

- The processor cannot afford to continually watch device to determine if it has some information and is ready to receive/transmit information.
- Polling



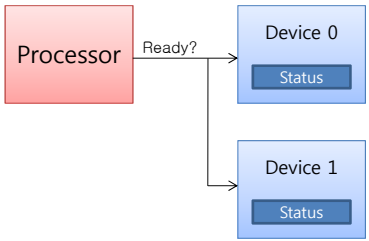
- Interrupt



Polling
(polled IO, software-driven IO)

- poll [poul] vt. ① (표를) 얻다.② 『+목+전+명』(표를) 던지다.③ 선거인 명부에 등록하다.④ …의 여론 조사를 하다.
- Refer to actively sampling the status of an external device as a synchronous activity
- Refers to the situation where a device is repeatedly checked for readiness.
- Simple, but inefficient
- Polling loop

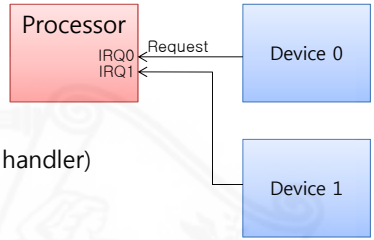
```
While (1)
{   if (*dev0_stat==1) serve0();
    if (*dev1_stat==1) serve1();
    …
}
```

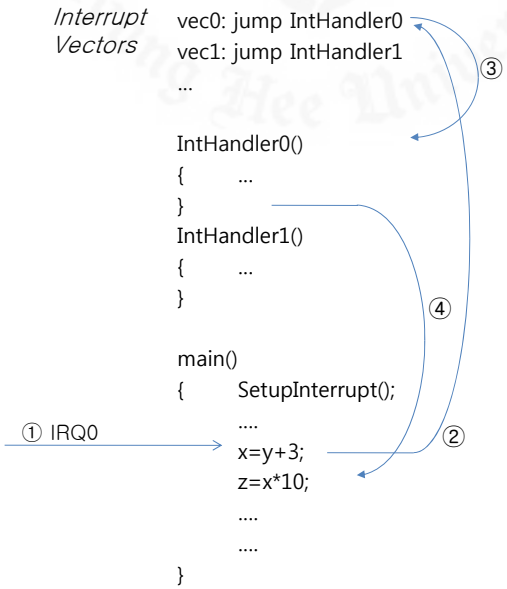


Interrupt

- interrupt [intərʌ́pt] vt. ① 가로막다, 저지하다, 훼방 놓다, (이야기 따위를) 중단시키다(in; during). ② (교통 따위를) 방해하다, 차단하다, 〖컴퓨터〗 가로채기하다.
- An asynchronous signal from hardware indicating the need for attention
- A way to avoid wasting the processor's valuable time in polling loops, waiting for external events.
- A hardware interrupt causes the processor to save its state of execution via a context switch, and begin execution of an interrupt handler.
- Efficient, but hardware is required.
- The program is not sequentially executed.

- IRQ (interrupt request)
  Maskable interrupt – IMR
  IACK (interrupt acknowlege)
- Interrupt vector
- Interrupt service routine (ISR; interrupt handler)



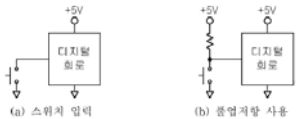

Interrupt Vectors
vec0: jump IntHandler0
vec1: jump IntHandler1
...
IntHandler0()
{ ...
}
IntHandler1()
{ ...
}
main()
{ SetupInterrupt();
....
x=y+3;
z=x*10;
....
....
}
① IRQ0
②
③
④

Pull-up/down Resistors

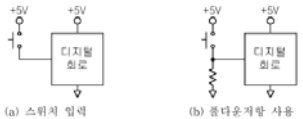
1. Input
   - 입력 논리값을 H,L 로 올바르게 인가하기 위해서

2. Output
   - Open collector 또는 Open drain 회로의 경우
   - 출력 전류를 증대시키려는 경우

3. 초기값을 정확하게 부여



<그림 1> L 스위치 입력과 풀업 저항



<그림 2> H 스위치 입력과 풀다운 저항