컴퓨터 구조 (Computer Architecture)

1. 컴퓨터 시스템

1) 구성요소
- 크게 중앙 처리 장치와 주변장치로 구분된다.
 ㄴ 중앙 처리 장치: CPU/MPU, 사물 인터넷 디바이스의 하드웨어 플랫폼 종류
 ㄴ 주변장치: 기억장치(메모리), 보조기억장치(저장장치), 입출력장치
- 메인보드에는 장치들의 인터페이스를 제공하며 데이터의 전달 통로가 디자인되어 있다.
- 네트워킹의 발전으로 컴퓨터 구조도 지속적으로 변화하고 있다.

2) 중앙 처리 장치 (Central Processing Unit)
- CPU: 실행 프로그램의 명령 해석하는 장치. 다른 장치의 제어, 연산, 기억을 담당한다.
 ㄴ ALU(산술논리 연산 장치), CU(제어 장치), 각종 레지스터(기억 장치)로 구성된다.
 ㄴ ALU(Arithmetic Logic Unit): 산술 논리 연산 장치. 산술 연산(사칙연산, 증감, 보수), 논리 연산(AND, OR, XOR, 시프트 연산)을 담당한다.
 ㄴ 부동소수는 FPU에서 연산하지만, 최근엔 ALU가 부동소수까지 처리하기도 한다.
- MPU(Micro Processor Unit): CPU를 고밀도 집적회로(LSI)화한 통합 장치
 ㄴ CISC(Complex Instruction Set Computer): 부팅 등 반복적인 기본 명령을 소프트웨어로 일일이 제어하지 않고 하드웨어로 제어하기 위해 많은 명령어 세트를 가지고 있는 컴퓨터
 ㄴ RISC(Reduced-): 실행속도를 높이기 위해 복잡한 처리는 소프트웨어에게 맡기도록 명령 세트를 축소 설계한 컴퓨터
 ㄴ Bit Slice MPU: MPU들을 각 사용자들이 원하는 요소별로 나누어 조합하는 방식
- 사물 인터넷(IOT) 디바이스 H/W 플랫폼: 가전제품이나 IOT기기 등에 사용된다.
 ㄴ Coretex-M3, Raspberry Pi, Galileo 등의 OS가 탑재된다.
 ㄴ ex) 아두이노(Arduino) - 대표적인 오픈소스 H/W 플랫폼

3) 주변장치 (Peripheral Device)
- RAM(Random Access Memory): 컴퓨터를 끄면 정보가 날아가는 휘발성 메모리
 ㄴ 주 기억장치: 넓은 의미로는 CPU의 레지스터 및 캐시들과 메모리 전체, 좁은 의미로는 DRAM(Dynamic RAM)만을 가리킨다.
- ROM(Read-Only Memory): 부팅 등의 주요 정보가 있는 읽기 전용 메모리
- 보조기억장치: RAM에 데이터를 적재하거나 RAM의 데이터를 저장하는 장치
 ㄴ 플래시 기억장치(EEPROM의 한 종류): 전기적인 충격을 통해 삭제가 가능한 형태의 비휘발성 메모리. 대표적인 예로 SSD가 있다.
 ㄴ 그 외에 SD카드나 USB 등이 있다.
- 입출력장치: 키보드, 마우스, 스캐너, 조이스틱, 바코드 판독기 등

4) 양자 컴퓨터
- 중첩(superposition), 얽힘(entanglement) 등 양자의 고유만 물리학적 특성을 이용하여, 다

수의 정보를 동시 처리할 수 있는 새로운 개념의 컴퓨터
- 현대 반도체 칩의 미세회로에서 발생하는 누설전류로 인한 고전적인 컴퓨터 구조의 한계 돌파를 위한 대안으로 대두되었다.
- 양자비트 또는 큐비트(qubit)를 정보처리 기본 단위로, 양자 병렬처리를 통해 연산속도를 기하급수적으로 증가시킬 수 있다.
 ㄴ 양자 병렬성(quantum parallelism): 여러 값의 연산을 동시에 수행하는 성질

* 웹은 사회성(유저 친화성)과 전문성(정보력)을 바탕으로 발전해왔다.

	사회성 낮음	사회성 높음
전문성 높음	Web 3.0 (Semantic) 지능형 웹 기술 (맞춤형 광고)	Web 4.0 (Ubiquitous) 현실과의 연결 (IOT, AI 등)
전문성 낮음	Web 1.0 초기 인터넷 (포털 사이트)	Web 2.0 (Social) 쌍방향 웹 기술 (위키, 커뮤니티)

 - 웹 기술의 발전에 따라 고성능의 컴퓨팅 시스템의 필요성도 증가해 왔다.

5) 소프트웨어 개발 패러다임
- 소프트웨어 개발 패러다임은 기존에 구조적 기법으로 사용하다가, 60~70년대 이후 지속적으로 변화해 왔다.
- 구조적 기법: 원하는 결과를 구하기 위해 일련의 함수로 프로그램을 작성한다.
 ㄴ 데이터와 함수는 분리되어 있다.
 ㄴ 개발과 유지보수에 많은 비용이 발생한다.
- 객체지향 기법(OOP): 데이터와 처리 기능(함수)을 '객체(Object)'로 단위화한 형태
 ㄴ 캡슐화: 연관된 데이터와 함수가 객체 단위로 묶여 있으며 필요한 부분만 노출한다.
 ㄴ 다형성: 각 기능의 상속으로 재사용이 가능하다.
- 컴포넌트 기법: 특정 기능을 가진 프로그램들을 조합시켜 외부와 인터페이스를 통해 동작하게 한다.
 ㄴ 집약성 있는 프로그램을 개별적인 동작 구현으로 분리, 시스템의 단순성을 높였다.
- 분산객체 프레임워크: 서로 다른 환경의 시스템에서 쉽게 상호작용할 수 있도록 하는 기술

2. 데이터의 표현

1) 정수 표현
- 기본적으로 데이터는 0과 1을 의미하는 비트(bit)로 구성된다.
 ㄴ 워드(WORD): CPU에서 취급하는 명령어나 데이터 길이. 8, 16, 32, 64bit 등 컴퓨터 시스템에 따라 다르다. (최근 컴퓨터는 32 또는 64)
- unpacked decimal: 각각의 숫자를 비트로 표현한다. 연산에 이용할 수 없고 입출력만 가능한 비수치 데이터.

ㄴ 예를 들어 95는 9와 5를 따로 표현하여 '1001 1000' 으로 나타낸다.
- packed decimal: 연산에 이용되지만 입출력은 불가능한 수치 데이터

2) 소수 표현
- 부동소수점(floating-point): 지수와 가수를 사용하여 소수점의 위치를 이동시킬 수 있는 표현 방법. 부호 1비트, 지수 8비트, 가수 23비트를 사용한다.
ㄴ (가수 * 밑수^지수) 로 나타내며, 밑수는 보통 2, 10, 16을 사용한다.
ㄴ 예를 들어, 0.4는 1.6 * 지수 2^{-2} 로 나타낼 수 있다.
- 배정밀도 부동소수점은 지수 11비트, 가수 52비트로 확장된다.
- 소수 부분은 2를 곱했을 때 자리수가 넘어가면(1이 넘으면) 1, 아니면 0이 된다.
ㄴ ex) 십진수 0.75의 경우, 2를 곱하면 1.5이므로 1이 붙고, 남은 0.5에 2를 곱하면 1이므로 다시 1이 붙는다. 즉 0.11이 된다.
ㄴ ex2) 십진수 0.8의 경우, 2를 곱하면 1.6이므로 1이 붙고, 남은 0.6에 2를 곱하면 1.2이므로 1이 붙고, 남은 0.2에 2를 곱하면 0.4, 0.8, 다시 1.6이 된다. 즉 0.11001100... 이다.

3) 보수 (Complement)
- 1의 보수(one's Complement): 해당 bit에서 가장 큰 표현형을 형성하는 데 서로 보완 관계에 있는 두 수의 관계.
ㄴ 십진수의 경우 더해서 9, 99, 999, ...가 되며 이진수는 더해서 11, 111, 1111, ... 등이 되는 수. 1010 - 0101 등 0과 1을 반대로 취한 수가 된다. (배타적 논리합은 항상 1111...)
- 2의 보수(two's Complement): 더하면 주어진 이진수보다 한 자릿수가 높은 최소값을 얻을 수 있는 두 수의 관계.
ㄴ 더해서 10, 100, 1000, ...이 되는 수. 0과 1을 반대로 바꾼 다음 1을 더해서 얻는다.

* 다른 사칙연산과 달리 뺄셈(A - B)은 2의 보수를 더하는 방식을 사용한다. (A + (~B+1))

3-1) 보수의 활용
- 패리티 비트: 오류가 생겼는지 검사하기 위해 정보에 덧붙이는 비트. 데이터의 비트에서 '1'의 개수에 따라 패리티 비트가 결정된다.
ㄴ 홀수 패리티 방식을 택한 경우, 1의 개수가 홀수면 패리티 비트를 0으로 한다.
ㄴ 에러 검출을 위해 데이터를 두 번 보내는 대신 패리티 비트를 사용한다.
- 부호로 활용: 첫 번째 비트가 1일 경우 음수를 나타낸다. 원래 데이터의 2의 보수를 표현한다.
ㄴ ex) byte 자료형 기준 십진수 54는 00110110, -54는 11001001 + 1인 11001010 이다.

4) 논리 연산
- 논리 게이트 (Logical gate): 2개 이상의 입력에 대해 AND, OR, XOR, NOT 등의 형식에 따라 하나의 출력값으로 나오는 논리 회로.
ㄴ 스위칭 이론: 직렬/병렬로 연결된 회로에 두 개의 스위치가 있을 때를 가정한 논리 연산
- 부울 대수: 참과 거짓을 판별할 수 있는 논리 명제를 식으로 표현한 것
ㄴ 논리 회로를 기술하는 데 사용되었다. 결합법칙, 분배법칙, 드모르간의 법칙이 적용된다.

- 카노(Karnaugh) 맵: 부울 대수를 이용한 식을 맵 형태로 간소화하여 표현한 것
 ㄴ 변수 개수 n개에 대한 경우의 수를 표로 나타내고, 각 항에 따라 값을 더해 나간다.
 ㄴ ex) a'b + abc + bc 의 계산 (단, a'는 not a를 의미함)

(1) a'b는 a'bc + a'bc' 로 나타낸다.

	bc	b'c	b'c'	bc'
a	0	0	0	0
a'	1	0	0	1

(2) 마찬가지로 abc와 bc도 표기한다. (중복되는 요소는 그대로 1)

	bc	b'c	b'c'	bc'
a	1	0	0	0
a'	1	0	0	1

(3) 가로 또는 세로줄에서 같은 요소를 찾아 추출한다.
abc+a'bc는 bc 로, a'bc+a'bc'는 a'b 로 추출된다. 최종적으로는 bc + a'b가 된다.

5) 조합 논리 회로
- 조합 회로: 입력과 출력을 가진 논리 게이트들의 집합.
- 순차 회로: 논리 게이트들 + 기억 능력이 있는 플립플롭으로 구성된다. 입력, 출력, 플립플롭의 상태(현재 값이 변하는가 유지되는가)를 통해 특정 지어진다.
- 플립플롭(Flip-Flop): 1비트의 정보를 저장 및 유지할 수 있는 회로. 작동 방식에 따라 SR, JK, D, T 플립플롭이 있다. 순차 회로 대부분에서 사용되며 캐시 메모리에도 사용된다.
 ㄴ SR(또는 RS): R-S 래치라고도 한다. R(Reset)과 S(Set)의 두 가지 입력값을 받으며, S가 1이면 1을, R이 1이면 0을 저장한다. 둘 다 0이면 현재 상태를 유지한다. 둘 다 1인 경우는 정의되어 있지 않다는 한계가 있다.
 ㄴ D: 입력값은 D 하나만 받는다. 입력받은 값이 현재값과 다르면 저장, 같으면 유지한다.
 ㄴ JK: 입력값은 J,K 두 가지를 받으며 각각이 S,R 역할을 하는 SR과 유사하나 둘 다 1 입력 시 현재 상태와 반대 값을 저장하는 동작이 추가되어 있다.
 ㄴ T: 입력값은 T 하나만 받는다. 0이면 현재 값을 유지하고, 1이면 현재 값의 반대가 된다.
- 멀티플렉서(Multiplexer): 다수의 입력 중 하나만 선별적으로 출력 가능하게 해주는 조합논리 회로
- 디멀티플렉서(De-): 하나의 입력을 다수의 출력으로 분해하는 기능의 조합회로

3. 중앙처리장치 (CPU)

1) 레지스터
- MAR(Memory Address Register): 메모리 주소 레지스터. 접근할 메모리 주소가 저장된다.
- MBR(Memory Buffer Register): 메모리 버퍼 레지스터. 데이터를 읽거나 쓸 때 MAR이 제공하는 주소의 데이터를 임시로 저장한다.
- IR(Instruction Register): 명령어 레지스터. PC가 가리키는 영역의 명령어를 저장한다. 해당 명령어 데이터는 MBR에서 읽어온다.
 ㄴ PC(Program Counter): 프로세스에서 현재 실행 중인 메모리 주소를 가리킨다. 순차적으

로 증가하지만, 프로그램의 분기문을 통해 PC값이 변경될 수 있다.
- 범용 레지스터(GR, GPR, General Perpose Register): 데이터 임시 저장, 주소 색인, 산술 및 논리 연산 등 다양하게 활용 가능한 레지스터
- 작업 레지스터(WR, working register): 산술논리연산을 실행할 수 있도록 자료를 저장하고 그 결과를 저장한다. ALU에 연결되어 있으며 프로그램으로 지정해서 사용할 수는 없다.
- 상태 레지스터(SR, Status Register): CPU 상태를 나타내는 특수 목적의 레지스터.
 ㄴ 연산 결과의 상태(현재 값이 중간값인지 최종 결과인지), 값의 부호(S), 오버플로우(V), 캐리 비트(C, 자리수 올림이 일어났는가), 인터럽트(I) 등을 표시한다.

1-1) 명령어 실행 예시
- ADD 명령을 실행한다고 가정한다.
 (1) PC가 가리키는 주소를 MAR에 저장하고, 해당 위치의 데이터가 MBR에 저장된다.
 (2) MBR의 데이터를 IR에서 읽어 어떤 명령어인지 확인한다.
 (3) IR의 주소 부분을 MAR로, ADD할 데이터를 MBR로 이동시킨다.
 (4) IR에서 지정한 내용을 GR에서 WR로 이동시킨다.
 (5) ADD 작업을 수행한다. WR의 데이터 + MBR의 데이터 => WR에 저장
 (6) 연산이 끝나면 WR의 결과를 GR로 이동하고, PC값을 1 증가시킨다.

2) 마이크로 명령
- 마이크로 연산(micro-operation): 하나의 클럭 펄스 내에서 실행되는 기본 동작
 ㄴ 시프트(shift), 카운트(count), 클리어(clear), 로드(road) 등이 있다.
- 마이크로 명령어 집합은 순차적 실행 명령어, 분기 명령어, 부 함수 호출 명령어, 복귀 명령어로 분류되며, 순차 실행 명령어가 전체의 70~80%를 차지한다.
- 명령어 구문 형식: 1형식(명령 코드+오퍼랜드 주소)과 2형식(명령 코드+주소 지정 모드+오퍼랜드 주소)의 두 가지가 있다.
 ㄴ 명령 코드: CPU에서 실행될 수 있는 연산
 ㄴ 오퍼랜드: 연산에 사용되는 값과, 그 값이 저장된 주소에 관한 정보
 ㄴ 주소 지정 모드: 오퍼랜드가 저장된 위치를 지정(인덱싱)하는 방식. 오퍼랜드 주소에 있는 값을 그대로 쓰면 되는 direct 방식과, 오퍼랜드 결과 값이 다시 다른 주소를 가리키는 indirect 방식이 있다.

2-1) 주소 지정 모드(addressing mode)
- 하드웨어와 소프웨어의 독립성을 유지하여 프로그램의 유연성을 가능케 하는 표준화 기법. 기계 중심의 프로그래밍이 가능해졌으며 소프트웨어는 포인터, 인덱서 등으로 주소를 가리킬 수 있다.
 ㄴ 명령어 집합은 CPU에선 기계 중심의 처리, 프로그램에선 자연어에 가까운 방식이 좋다.

설계 관점	기준	자연어에 가까운 명령 코드	기계 중심의 명령 코드
프로그램	프로그래밍 난이도	쉬움	어려움
	프로그램 길이	짧음	길
	번역기 설계 난이도	쉬움	어려움
CPU 구조	표준화 용이성	어려움	쉬움
	명령어 길이	길	짧음
	제어장치(ALU,레지스터 등)의 제어 난이도	복잡함	용이함

- 묵시적 모드: 오퍼랜드에 포함되지 않은 특수 모드
 ㄴ NOP(No Operation): 오퍼랜드가 필요없는 명령어
 ㄴ INC: 묵시적 오퍼랜드인 누산기(AC, accumulator)의 연산 명령어. 누산기는 계산 결과 값을 임시로 저장하는 레지스터.
 ㄴ ADD: 스택 구조의 명령어. 스택에 오퍼랜드가 저장된다.
- 직접 값 모드: 오퍼랜드 자체가 명령어에 포함되어 있는 모드
 ㄴ ex) MOV R1, #100 : 십진수 100이 두 번째 오퍼랜드로 직접 포함되어 있다.
- 레지스터 모드: 오퍼랜드가 레지스터에 저장된 모드
 ㄴ ex) ADD R1,R2 : 레지스터의 값 R1과 R2에 보유한 값을 오퍼랜드로 사용한다.
- 메모리 직접 주소 모드(direct mode): 오퍼랜드가 저장된 메모리 주소를 나타내는 모드
 ㄴ ex) MOV R1, 100 : 메모리의 100번지 내용으로 이동하라는 내용. 해당 번지의 데이터가 오퍼랜드가 된다. (R1과 100이 바뀐 형태도 가능)
- 메모리 간접 주소 모드(indiect mode): 메모리를 이용해 간접적으로 주소를 지정하는 모드
 ㄴ ex) MOV R1, @100 : 메모리의 100번지 내용에 주소가 기입되어 있다는 내용. 100번지의 주소를 다시 찾아가야 실제 유효 값을 찾을 수 있다.

* MOV a,b 는 a에 b의 값을 저장하는 명령이다.

3) 입출력
- 입출력은 인터페이스를 통해 외부 장치와 CPU간의 정보를 전송하는 형태로 구성된다.
- [입출력장치 - 인터페이스 - 레지스터] 의 정보 전송은 직렬로 이루어지며, 레지스터에서 정보를 주고받는 과정은 병렬로 처리된다.
 ㄴ 즉 키보드 입력 시 레지스터에서 1비트씩 정보를 입력받아 8비트가 되면 한 번에 병렬로 내보낸다.
 ㄴ 병렬 연결은 제어 플립플롭 FGI(Input), FGO(Output)라는 플래그 비트로 제어된다. 레지스터에 정보가 들어오는 도중에 내보내지 않도록, 모든 비트가 채워진 후 플래그가 1이 되면 정보가 전송된다.
- 입력 예시: 키보드 입력 -> 입력 인터페이스 -> INPR(Input Register) -> AC(누산기)
- 출력 예시: AC -> OUTR(Output Register) -> 출력 인터페이스 -> 프린트 출력

3-1) 인터럽트
- R 플립플롭에서 1을 반환할 경우 인터럽트 사이클로 넘어간다.

- 인터럽트 발생 시 현재 작업중인 PC(프로그램 카운터)를 메모리 0번지에 저장하고 PC가 1로 바뀐다. 인터럽트 처리를 한 후에 IEN과 R을 0으로 되돌리고 다음 사이클로 넘어간다.
 ㄴ 인터럽트로 인한 프로세스 처리 후 다시 해당 지점으로 돌아오기 위해서다.
 ㄴ IEN(Interrupt Enable flip-flop): 컴퓨터에 인터럽트 신호를 전달할지 말지 결정하는 플립플롭. 1일 경우 플래그를 확인하여 인터럽트를 건다.
- 입출력장치의 처리속도가 CPU에 비해 수천 배 이상 느리므로, 입출력 플래그(FGI, FGO)를 이용한 방식으로 감지하는 것은 매우 비효율적이라 IEN을 이용한다. 평상시엔 플래그를 이용하지 않고, 외부장치에 의해 IEN이 1이 되면 FGI, FGO 플래그를 체크한다.

4) 프로그래밍
- 소프트웨어는 크게 시스템 소프트웨어와 응용 소프트웨어로 나뉜다.
 ㄴ 시스템 소프트웨어: OS(스케쥴러, 네트워크 관리 등), 언어번역(컴파일러, 어셈블러, 인터프리터 등), 유틸리티(백신, 드라이버 관리, DB 관리 시스템)
 ㄴ 응용 소프트웨어(Application): 각종 사무용 프로그램, 멀티미디어, 게임, 메신저 등
- 어셈블리어: 기계어의 한 동작을 1:1로 대응시켜 표현한 언어. 컴퓨터마다 어셈블리어를 가지고 있으며 제조업체에서 지정한 규칙을 따라야 한다. 라벨과 명령어로 구성된다.
 ㄴ 라벨 필드: 3자 이하의 문자, 숫자. 첫 문자는 반드시 문자여야 한다. 라벨 필드가 비어있는 경우도 있다.
 ㄴ 명령어 필드: 기계 명령어나 수도(pseudo) 명령어를 작성한다. 수도 명령은 실제 기능이 있는 명령이 아닌 것으로, 예시로는 다음 명령의 시작 위치를 알려주는 'ORG'가 있다.
 ㄴ 코멘트 필드: 주석. 불필요한 경우 생략.

4. 파이프라인

1) 병렬처리(parallel processing)
- 데이터를 동시에 처리하는 기능을 제공하는 기술 전반을 가리키는 개념
 ㄴ 동시 전송이 가능한 병렬성을 갖는 레지스터 활용하거나, 데이터를 여러 개의 장치에 분산시키는 등의 방식을 사용한다.
 ㄴ 하나의 계산 결과를 통해 다음 연산을 수행해야 하는 상황에서는 문제가 생길 수 있다.
- 마이클 플린(M.J Flynn)은 병렬처리를 SISD, SIMD, MISD, MIMD의 네 가지로 분류했다. 예를 들어 SIMD는 단일 명령어(Instruction) + 다중 데이터 스트림 구조를 의미한다.
 ㄴ SISD: 단일 컴퓨터 구조. 명령어는 순차적으로 실행되며, 병렬처리는 다중 기능 장치나 파이프라인 처리에 의해 구현된다.
 ㄴ SIMD: 공통의 제어장치 아래에 여러 개의 처리 장치를 두는 구조. 하나의 명령을 여러 개의 데이터에 적용시켜 동시에 계산할 수 있다. 여러 프로세서가 동시에 접근할 수 있는 공유 메모리 장치가 필요하다. 벡터나 배열의 계산에 사용된다.
 ㄴ MISD: 다중 명령어에 단일 데이터. 이론적으로만 연구되고 있다.
 ㄴ MIMD: 여러 프로그램을 동시에 수행하는 시스템. 대부분의 멀티 프로세서와 다중 컴퓨팅 시스템이 여기에 속한다.
 ㄴ 이는 이론적 고찰이라기보다 외양적 행동 양상에 따른 분류일 뿐이다. 대표적으로 파이프

라인은 이 분류 방식에 적합하지 않다.

2) 파이프라인 구조
- 하나의 프로세서를 여러 개의 서브 프로세스로 나누어서, 각 프로세스가 동시에 다른 데이터를 처리하게 하는 방식
 ㄴ 동일한 작업을 다른 데이터에 적용시켜 여러 번 반복할 때 효과적이다.
 ㄴ 동일한 복잡도를 가진 부연산들로 나누어지는 연산 동작이라면 무엇이든 파이프라인 프로세서에 의해 구현될 수 있다. (작업량이 서로 다른 경우는 느린 쪽에 맞춰지니 부적합함)
- 프로세스를 세그먼트(segment) 단위로 나누어 진행한다. 마지막 세그먼트를 통과하면 최종적인 연산 결과가 도출된다. 각 세그먼트마다 레지스터가 존재한다.
 ㄴ 중간 결과를 각각의 레지스터에 저장하고, 클럭 펄스 신호를 받아 처리할 기술로 인해 도입되었다.
 ㄴ ex) (A+B-C) 계산 -> 'A입력(R1), B입력(R2), R1+R2(R3), C입력(R4), R3+R4(R5)'의 다섯 가지 세그먼트로 구분
- 각 세그먼트는 정해진 부 연산 처리 후 중간 결과를 각각의 레지스터에 저장한다. 공통으로 받는 클럭 제어에 의해 정보가 다음 세그먼트로 이동한다.
- 데이터 하나를 처리하는 데 k 세그먼트가 필요하고, 처리할 데이터 개수가 n개일 경우, 비파이프라인은 (k * n) 만큼의 시간이 소요되지만, 파이프라인은 k + (n-1) 만큼 소요된다.
 ㄴ ex) 4세그먼트로 나눌 수 있는 연산을 100개의 데이터에 대해 처리할 경우, 파이프라인 구조에서는 4 + (100-1) = 103세그먼트 만큼 시간이 소요된다. (1세그먼트당 10ns가 걸린면 총 1030ns 소요. 비 파이프라인은 400 * 10 = 4000ns 소요로 약 4배.)
 ㄴ n이 무한히 많아질 경우, 이론상 세그먼트 수만큼 효율이 배로 증가한다.
- 현실적으로 이론상 최대 속도를 구현하지는 못 한다.
 ㄴ 각 세그먼트가 부연산을 수행하는 시간이 서로 다르다.
 ㄴ 레지스터를 제어하는 클럭 사이클은 가장 느린 세그먼트에 싱크를 맞추어야 한다.
 ㄴ 그럼에도 순차적인 작업을 나누어 동시에 진행할 수 있으므로, 각 태스크를 한번에 수행하는 병렬적인 다중 기능 장치(SIMD 구조) 등에 비해서는 훨씬 효율적이다.

* 한 명령어가 1-2-3-4단계를 거쳐 완료되고, 해당 명령을 A,B,C,D의 네 가지 데이터에 대해 수행해야 할 경우
 - 비 파이프라인: A1 - A2 - A3 - ... - D3 - D4 의 16 단계
 - 파이프라인: 프로세스를 4개의 세그먼트로 나누어 개별적으로 처리한다. 총 7 단계

	1단계	2단계	3단계	4단계	5단계	6단계	7단계
세그먼트1	A1	B1	C1	D1			
세그먼트2		A2	B2	C2	D2		
세그먼트3			A3	B3	C3	D3	
세그먼트4				A4	B4	C4	D4

3) 명령어 파이프라인
- 산술 파이프라인: 산술 연산(부동/고정 소수점 포함)을 부연산으로 나누어 처리
- 명령어 파이프라인: 명령어 사이클의 fetch, decode, execute 단계를 중첩시켜 처리

ㄴ 일반적으로 컴퓨터 명령어는 '명령어 인출(fetch), 해석(decode), 피연산자 인출, 수행(execute)'의 4단계를 거친다.
- 명령어 파이프라인은 이전 명령어가 실행되는 동안 다음 명령어를 읽어와 동시에 수행한다.
ㄴ 이 특성상 분기가 발생하면 현재 파이프라인은 모두 비워지고, 메모리에 읽어온 명령어 중 분기점 이후의 명령은 모두 무시되어야 한다.
ㄴ 많은 분기문은 파이프라인 시스템을 채택한 컴퓨터 성능 저하의 주 요인이 된다.

* 분기가 일어날 경우 시간별 파이프라인 처리
- 각 명령어는 FI(Fetch Instruction), D(Decode), FO(Fetch Operand), E(Execute)의 4단계로 나누어지며, 명령2의 FI 이후 이 명령이 분기 명령임을 확인했다고 가정한다.

	1단계	2	3	4	5	6	7	8	9	10	11
명령1	FI	D	FO	E							
2		FI	D	FO	E						
3			~~FI~~	-	-	FI	D	FO	E		
4				-	-	-	FI	D	FO	E	
5								FI	D	FO	E

- 2단계에서 분기를 확인하여 3단계의 FI 명령은 무시되고, 이후엔 명령을 받지 않는다.

3-1) 파이프라인 분기 예측
- 분기 예측(branch prediction): 분기 및 적재 명령어가 반복 참조하는 오퍼랜드를 신속하게 제공한다. 자주, 규칙적, 반복적으로 사용되는 명령이 정해져 있다는 점에서 착안했다.
ㄴ 예측 실패 시에는 오히려 시간이 지연될 수 있다.
- 명령어 파이프라인은 정상 동작에서 이탈하는 요인들을 고려하여 만들어야 한다.
(1) 자원 충돌(Resources Conflict): 두 세그먼트가 동시에 메모리 접근
-> 명령어 메모리와 데이터 메모리 분리하여 해결
(2) 데이터 의존성(Data Dependency): 이전 명령어 결과에 의존하여 다음 명령어가 실행되어야 하는데, 이전 명령어 결과값이 아직 준비되지 않아서 충돌이 발생
-> 하드웨어 인터락(Hardware Interlock): 명령어의 피연산자가 앞서간 명령어의 목적지와 일치하는 경우, 충돌을 피할 수 있도록 피연산자가 준비되지 않은 명령어 실행을 지연시킨다.
-> 오퍼랜드 포워딩(Operand Forwarding): 충돌이 예상되는 지점이 있을 경우, 정해진 순서가 아닌 별도의 순서에 따라 직접 파이프라인 세그먼트에 오퍼랜드를 전달한다.
(3) 주소 의존성(Address-): 레지스터 간접 모드 사용 시, 메모리에서 주소를 로드하여 다시 그 주소의 값을 찾아가야 하는 상황이라면 곧바로 피연산자를 fetch하지 못 하고 대기해야 함
(4) 분기 곤란(Branch Difficulty): 분기 명령어 등 프로그램 카운터(PC) 값을 변경시키는 명령어에 의해 발생한다.
-> 분기 목표 버퍼(BTB, Branch Target Buffer)를 사용한다. 분기 명령어와 분기 목표 명령어를 캐싱해 두고, 동일한 분기문에 대해 다시 해당 분기 목표로 갈 것이라 예측한다.
ㄴ 예를 들어 50line의 if문이 78line으로 향했다면 해당 정보를 저장, 다음에 50line을 만날 경우 BTB에서 78line을 탐색한다. 예측이 틀릴 경우는 BTB를 지우고 원래대로 처리.

4) 파이프라인 CPU
- RISC 프로세서: 명령 세트를 축소하여 실행속도를 높인 컴퓨터(1-2) 참고). 명령어 수는 증가하지만 명령어당 실행 클록 수(CPI)와 파이프라인 구조를 이용해 작업시간을 줄인다.
- RISC 프로세서는 파이프라인에 최적화된 구조를 가지고 있다.
 ㄴ 간단한 명령 코드와 주소 지정 모드
 ㄴ CPU 내의 캐시 메모리, 신속한 오퍼랜드 참조
 ㄴ 문맥 전환에 유리한 중첩된 레지스터 윈도우
 ㄴ 실수 연산을 별도로 처리하기 위한 코프로세서(Co-Processor)
- RISC 파이프 라인: 명령어를 I, A, E의 3개의 세그먼트로 나눈다.
 ㄴ 명령어의 fetch(I), ALU의 동작(A), 명령어 실행(E)
- 이론상으로는 세그먼트 횟수가 많아질수록 최대속도가 증가하지만, 파이프라인 시스템이 최고 효율을 내기 위해서는 아래 조건이 만족되어야 한다.
 (1) 모든 명령어는 세그먼트가 적용된 동일한 처리 과정으로 처리되어야 한다.
 (2) 각 세그먼트의 처리 시간이 일정해야 한다.
 (3) 명령어는 순차적으로 실행되어야 한다.
  ㄴ 분기 명령, 함수 호출, 반환 등에 의해 순차성이 깨진다.
 (4) 사용 명령어 간 상호 의존성이 없어야 하며, 공유 자원의 충돌이 없어야 한다.

5. 메모리 구조

1) 메모리 계층과 특성
- 주 기억장치(Main Memory Unit)
 ㄴ DRAM: 비교적 대용량, 주기적으로 재충전해 주어야 정보가 유지, 전력 소비 적음
 ㄴ ROM도 주 기억장치의 일종
- 캐시 메모리(Cache Memory, buffer)
 ㄴ SRAM: 전원이 연결된 동안 정보 유지, Filp-Flop으로 구성. 사용이 편리하고 빠르다.
 ㄴ 프로세스 논리회로가 주기억장치 접근 시간보다 빨라서 고안된 프로세서 내 메모리
 ㄴ 실행중인 프로세스 일부 또는 사용빈도가 높은 임시 데이터 저장
- 보조 기억장치(Auxiliary Memory Unit)
 ㄴ 비교적 저속, 비휘발성, 대용량의 저장장치
 ㄴ CPU가 직접 컨트롤하지 않으며, I/O Processor에 의해 주 기억장치로 옮겨져야(Loading) 실행이 가능하다.
- 메모리는 CS(칩 선택), RD(읽기), WR(쓰기)의 3가지 신호를 받는다. CS가 0이면 작동하지 않고, CS가 1일 때 RD 또는 WR 신호가 켜져 있는지(=1) 감지하여 읽기나 쓰기를 수행한다.
- 접근 시간: 보조기억장치 내 기억장소에 도달하여 그 내용을 얻는 데 요구되는 평균 시간
 ㄴ 시크 타임(Seek time): 헤더(read/write)가 지정된 장소에 도달하는 데 소요되는 시간
 ㄴ 트랜스퍼 타임(Transfer time): 추출된 데이터를 장치내/외의 필요한 곳으로 전송하는데 요구되는 시간
 ㄴ 레코드(record): 데이터가 기록되는 단위(길이), 시크 타임을 체크하는 기준
 ㄴ 전송률: 장치가 레코드 시작점에서 단위 시간(보통 1분)동안 전송 가능한 문자 수

* 메모리의 상대적인 속도와 가격
 - Static RAM(SRAM): 0.5~2.5ns / 1GB당 $2000~5000
 - Dynamic RAM(DRAM): 50~70ns / 1GB당 $20~75
 - Magnetic Disk: 5~20ms (5000~20000ns) / 1GB당 $0.2~2

2) 메모리 관리 정책
- 연관 기억장치(Associative Memory): 주소 대신 내용을 통해 정보가 저장된 위치를 찾을 수 있는 기억장치. 내용 지정(content addressable) 메모리라고도 한다.
 ㄴ 데이터의 주소가 아닌 내용을 통해 병렬 탐색하므로, 빠른 속도의 탐색이 가능하다. 탐색은 전체 워드 또는 일부 내용으로 실행될 수 있다.
 ㄴ 인자 워드(Argument Register): 찾으려는 값을 넣는다.
 ㄴ Key Register: 마스크 비트를 통해 특정 비트만 빠르게 비교하도록 입력 내용을 마스킹한다.
 ㄴ Match Register: 인자 워드와 동일한 내용 발견 시 1로 체크된다.
 ㄴ 내용을 비교할 수 있는 논리회로가 필요하므로 주기억장치(DRAM)보다 비싸다.
- 캐시 메모리(Cache Memory): 참조의 국한성에서 착안, 빈번하게 참조되는 메모리 내용을 저장하여 CPU 처리속도와 근접하는 수준으로 빠르게 접근하기 위한 기억장치.
 ㄴ 참조의 국한성(locality of reference): 프로그램이 실행되는 동안, 메모리 참조는 특정 영역에서만 이루어지는 경향이 있다.
 ㄴ CPU는 메모리 접근 시 캐시를 체크한다. 워드가 캐시에 있으면(hit) 해당 블록(1~16워드)을 읽어오고, 아니면(miss) 주기억장치에 접근한다. 통상적으로 히트율은 90%에 근접한다.

3) 메모리 매핑 기법
- 캐시 메모리는 자주 참조될 내용을 효과적으로 매핑하여 캐시를 구성하는 것이 중요하다. 이를 위한 매핑 방법론으로 Associate, Direct, Set-Associate 등의 방식이 있다.
- 기본적으로 캐시 미스 시 해당 데이터 주소는 캐시에 새로 적재된다. 캐시와 주기억장치 접근 속도는 약 7배 수준으로, 미스율을 최대한 낮추는 것이 중요하다.
- 연관사상(Associate Mapping): 주기억장치의 데이터가 캐시 메모리 어느 곳이든 적재될 수 있다.
 ㄴ 가장 빠르고 유연하며 안정적이다. 일부 내용들을 반복 참조 시 히트율이 높지만, 가격이 비싸고 구현 비용도 높다.
 ㄴ 캐시가 가득 차 있다면 기존 캐시들 중 주어진 알고리즘에 의해 하나가 새로운 주소로 대체된다. (대체될 데이터를 랜덤으로 결정하는 경우도 있다.)
- 직접사상(Direct Mapping): 주기억장치의 블록이 캐시의 특정 라인에 1:1 대응되어, 데이터는 지정된 라인에만 적재가 가능하다.
 ㄴ 캐시 탐색 시 지정된 라인만 보면 캐시 적중(hit) 여부를 알 수 있다. 단순하고 저렴하지만 특정 영역의 데이터를 캐시에 적재할 수 있는 양이 제한적이라 캐시 미스율이 높다.
 ㄴ 지정된 영역의 캐시가 차 있다면 해당 라인의 내용이 새로운 내용으로 대체된다.
- 집합 연관사상(Set-Associate Mapping): 주기억장치 블록과 캐시의 라인들을 집합(set)으로 묶어 대응시킨다. 데이터는 캐시 메모리의 지정된 일부 라인들에 적재될 수 있다.

ㄴ 위 두 가지의 장점을 합쳐서 개선한 형태
ㄴ 지정된 영역의 캐시가 차 있다면 해당 라인 내에서 특정 데이터가 대체된다.

* 가상 메모리 시스템과의 비교

구분	가상 메모리 시스템	캐시 메모리 시스템
위치	보조기억장치와 주기억장치 사이의 데이터 전송 관리	주기억장치와 CPU 사이의 정보 교환
필요성	메모리 공간 절약	메모리 접근 비용 절약
사용 목적	사용되지 않는 메모리 영역 관리	사용 빈도가 높은 데이터 저장/활용
활용 기법	페이지 테이블, 페이지 폴트	직접사상, 연관사상, 집합 연관사상

4) 메모리 관리 시스템
- 프로그램 간 데이터 흐름, 선후 데이터 활용, 메모리 사용량 조절, 다른 프로그램에 영향을 끼치지 않게 하는 등 메모리를 관리하는 시스템
- 메모리 관리 하드웨어와 OS의 일부에 속하는 메모리 관리 소프트웨어가 있다.
ㄴ 메모리 관리 하드웨어: 논리 메모리 참조-물리 메모리 주소의 변환, 프로그램과 데이터의 저장 공간 분리, 서로 다른 사용자가 한 프로그램을 같이 사용하지 위한 편의, 사용자의 허락되지 않은 접근과 OS 변형 방지 및 정보 보호 등
- 메모리의 광역화(가상 메모리, 캐시 메모리), 멀티 프로그램(파이프라인) 등의 발달로 장점도 있으나, 여러 시스템 간의 상호 간섭도 시스템 기능 저하의 원인이 된다.

5) 다양한 기억장치들
- DDR: RAM 규격의 일종으로 1997년 삼성에 의해 발표되었다.
- SDRAM(Synchronous DRAM): 동기식 DRAM. 클럭 속도가 CPU와 동기화되어 있는 DRAM 종류를 가리킨다. CPU의 클럭 펄스가 1번 변할 때 한 차례의 정보 전송을 허용한다.
ㄴ CPU와 함께 움직여서 CPU가 수행할 수 있는 명령어 개수 증가에 도움을 준다.
- DDR(Doudle Data Rate) SDRAM: 클럭 신호가 상승 및 하강할 때 각각 한 차례씩 데이터를 전송한다.
ㄴ SDR(Single Data Rate) 대비 클럭 주파수 증가 없이 전송 속도 2배 향상
ㄴ DDR2,3,4 등으로 발전할수록 높은 전송 속도와 낮은 전력 소비를 제공한다.
ㄴ DDR2: 내부 클럭 속도는 같으나 향상된 I/O 버스 신호로 전송 속도가 증가했다.
ㄴ DDR3: ASR(Automatic Self-Refresh), SRT(Self-Refresh Temperature) 등 제공
ㄴ DDR4: DBI(Data Bus Inversion), CRC(Cyclic Redundancy Check), CA parity 등의 기능 추가로 신호 무결성을 향상, 데이터 전송 및 액세스의 안정성을 향상시켰다.
- SSD(Solid State Drive): 고형 상태 보조기억장치. 플래시 메모리로 구성된다.
ㄴ 자기 디스크인 HDD와 달리 구동부(모터)가 없어 소음, 전역, 발열 모두 낮다.
ㄴ HDD와의 관계를 출력장치로 비유하면 액정 디스플레이(LCD)와 형상 기억 장치를 사용하는 AMOLED의 차이와 유사하다.
- RAID(Redundant Array of Inexpensive/Independent Disks): 성능향상을 위해 저렴하고 크기가 작은 여러 개의 하드 디스크를 묶어 하나의 기억장치처럼 사용하는 방식.

ㄴ 오류 복구 정책에 따라 RAID 0~6 레벨로 구분된다. 낮을수록 안정성이 떨어진다.
- USB(Universal Serial Bus): 주변기기 연결 인터페이스에서 독점적인 위치를 점하고 있다.

6. 입출력 구조

1) 버스 시스템
- 버스(bus): 컴퓨터 내부에서 신호를 주고받는 통로
 ㄴ 전선 혹은 PCB(Printed Circuit Board) 형태로 구현된 물리적 장치
- 버스 시스템: 레지스터들 사이의 정보 전송 경로를 번거롭게 연결하는 대신, 공통의 버스 라인(Common bus line)에 연결해 제어하는 것
 ㄴ 시스템 버스, I/O 버스 등 관련성이 있는 장치들을 모아서 관리한다.
- I/O 버스 라인에 여러 개의 입출력 제어기가 연결되고, I/O 버스 라인은 버스 어댑터에 의해 시스템 버스의 CPU/메모리와 통신한다.
 ㄴ 버스 어댑터(Bus Adapter): 입출력 버스와 시스템 버스 라인을 연결하는 장치
 ㄴ 입출력 제어기(I/O controller): 입출력 인터페이스
- 컴퓨터 시스템에는 다양한 버스가 존재한다.
 ㄴ 시스템 버스 내에 데이터, 주소, 읽기/쓰기 정보를 전송하는 메모리 버스가 있다.
 ㄴ CPU 내부에도 레지스터와 ALU 사이의 정보 전송 등을 위한 내부 버스가 있다.
 ㄴ 멀티 프로세서 시스템은 시스템 버스를 통해 공유 메모리를 관리한다. (임계 구역 제어와 중재 등 포함)

2) 시스템 버스 제어
- 버스 라인을 디자인할 때, 시스템 버스와 I/O 버스 간 상호 중재는 IEEE 표준 796 다중 버스 신호를 따른다.
 ㄴ 데이터 라인, 주소 라인, 제어 라인으로 구성된다.
- 데이터 라인: 프로세스와 공용 메모리 사이의 전송 경로를 제공한다. (양방향 정보 전송, 32line이 가장 일반적)
 ㄴ 데이터 전송 모드는 동기/비동기 방식이 있다.
 ㄴ 동기: source-target 장치 간 공통 클럭에 의해 데이터가 전송된다.
 ㄴ 비동기: 각 장치들이 독립적인 클럭으로 작동하며, source-target 장치는 핸드 셰이킹(hand shaking) 제어신호에 의해 데이터를 전송한다.
- 주소 라인: 메모리 주소와 입출력 포트 식별에 활용된다. (단방향 정보 전송)
- 제어 라인: 장치들 간의 정보 전송을 제어하는 신호를 제공한다. 메모리/입출력 장치의 읽기와 쓰기, 인터럽트 제어, 버스 제어, 기타 여러 가지 신호들로 구성된다.
 ㄴ 전송: 메모리 읽기와 쓰기 등
 ㄴ 전송 승낙: 전송이 완료되었음을 알림
 ㄴ 인터럽트 요구: 8개의 인터럽트 요구 명령과 1개의 승낙 명령으로 구성. 우선 순위 인터럽트 제어기에 연결되어 활용된다.
 ㄴ 버스 제어 신호: 버스 요구, 버스 승인
 ㄴ 그 외에 버스 락(bus lock), 중재 절차를 위한 신호 등

- 시스템 버스 제어기는 버스 중재를 위한 논리 회로가 내장되어 있다. 크게 직렬/병렬 중재 방식으로 구분된다.

2-1) 직렬 중재 절차 (Serial)
- 우선순위를 바탕으로 데이지 체인(Daisy-chain) 연결로 중재하는 방식
 ㄴ 데이지 체인: CPU에 전기적으로 가까운 장치가 우선권을 갖는 방식의 연결
- 각 중재자는 PI(Priority Input), PO(Priority Output)을 가진다. 버스 사용을 원치 않을 경우 PI=0, PO=1이 되어 다음 중재자에게 전송된다.
 ㄴ 가장 우선순위가 높은 중재자의 PI는 항상 1, PO는 항상 0이다.
 ㄴ PI가 1인 중재자는 신호를 받아서, 자신이 사용할 것이라면 PO를 0으로 하고 버스를 요청한다. 사용하지 않으면 PO를 1로 하여 신호를 다음 중재자에게 전송한다.
- PI=1, PO=0을 할당받은 중재자는 버스 신청 전 Bus busy line을 확인한다.
 ㄴ Bus busy line: 버스를 사용 중일 경우 활성화된다.
 ㄴ 비활성 상태일 경우 해당 프로세서는 버스 라인을 할당받고 Bus busy line을 활성 상태로 변경시킨다.
 ㄴ 활성 상태일 경우 나보다 낮은 우선순위의 프로세서가 이미 버스를 사용 중이라는 의미이므로, 인터럽트가 아닌 이상 끝날 때까지 대기해야 한다.

2-2) 병렬 중재 절차 (Parallel)
- 각 버스 중재자는 버스 요청 출력 라인(Req)과 버스 승낙 입력 라인(Ack)을 가진다.
 ㄴ 각 중재자에게 요청을 받아서 인코딩 - 디코딩을 거쳐 승낙 여부를 반환한다.
 ㄴ 인코딩에는 각 중재 요청에 따른 output을 정리한 인코더 진리표가 사용된다.
- 버스 비지 라인은 Daisy-chain 연결 때와 동일
 ㄴ 활성 상태면 버스가 사용 중임을 모든 중재자에게 알린다.

3) 입출력 연결
- 입출력 주소 지정: 다양한 장치들을 구분하기 위해, 각각의 입출력 장치 및 통신 포트에 고유 주소를 할당한다.
 ㄴ 하나의 입출력 장치에 그 상태나 용도(입력/출력)에 따라 다수의 주소 할당이 가능하다.
 (1) 메모리 맵 입출력: 메모리의 주소 공간 일부를 입출력 주소 공간으로 활용
 - 동일한 주소선과 제어선으로 입출력 관리 가능
 - 주소 값에 따라 데이터 저장 공간과 I/O장치를 구분하므로 이와 관련된 구현이 필요하다.
  ㄴ 이는 현재 모든 CPU에 구현되어 있으므로, 어느 시스템에서나 사용 가능하다는 장점이 있다.
 (2) 입출력 맵 입출력: 메모리 주소 공간과 독립적으로 구분된 입출력 주소 공간을 할당
 - 메모리와 입출력 주소를 구분하는 제어 선을 사용
 - 독립된 공간을 가지므로 주소가 동일한 경우 명령어를 구분할 수 있어야 한다. 즉 제어 신호를 통해 메모리 또는 입출력 장치에 접근하도록 하드웨어적 구현이 필요하다.
  ㄴ '중구'가 전달될 때 부산시 중구인지 서울시 중구인지 구분하는 것과 비슷하다.
  ㄴ 따라서 별도의 I/O용 명령어와 이를 위한 외부 제어선이 있는 CPU만 구현 가능하다.
  ㄴ 일반적인 시중의 컴퓨터들은 입출력 맵 입출력을 하드웨어상 지원한다.

- 입출력 버스 할당: 제어기에 의해 입출력 버스의 할당이 이루어진다. 위의 직렬/병렬 중재를 제어기(controller)의 관점에서 보면, 중앙 제어와 분산 제어로 나눌 수 있다.
ㄴ I/O 버스는 주로 중앙 제어, 시스템 버스는 분산 제어 시스템을 선호한다.
(1) 중앙 제어 버스: 병렬 처리와 유사하다. 버스 요청을 제어기 하나를 통해 제어한다.
- 버스에 연결된 장치들이 제어기에 요청을 보내고, 제어기는 버스가 사용 중인지 확인하여 버스 승낙 신호를 보낸다.
- 구현 및 관리는 용이하지만, 제어기에 이상이 생기면 버스 중재에 큰 결함이 생긴다.
(2) 분산 제어 버스: 직렬 처리와 유사하다. 각 장치가 제어 기능을 나누어 실행한다.
- 중재자(장치) 하나하나의 구현 비용이 높으나 문제 상황에 유연하다.
- 입출력 수행
(1) 동기 버스에 의한 수행
- 버스에 연결된 모든 장치에 동일하게 적용되는 버스 클럭(clock)이 있다.
- 클럭이 0 -> 1이 될 때 주소 및 제어버스에서 입출력이 필요한 장치에 신호를 발송한다.
- 장치는 주소를 식별한 후, 클럭이 1 -> 0이 될 때 데이터 전송을 시작한다.
- 별도의 동기화용 신호가 불필요한 대신, 모든 장치는 클럭에 맞추어 입출력이 진행되도록 설계되어야 한다.
(2) 비동기 버스에 의한 수행
- 일종의 핸드 셰이크 신호를 이용하여 필요할 때 장치를 호출한다.
- 주소선에서 지정된 입출력 장치에 주소 전송하면, CPU가 I/O controller를 통해 장치에 Ready 신호를 보내고, 장치는 Accept 신호로 응답을 보낸 후 데이터를 전송한다.
- 데이터 전송이 완료되면 작업 종료 신호를 보낸다.
- 융통성 있는 시스템을 구성할 수 있으며 입출력이 자주 일어나지 않을 경우 특히 효율적이지만, 프로그램 작동 방식이 복잡해진다.

3-1) 동적 중재 알고리즘
- 시스템이 동작하는 동안 장치들의 우선순위를 변경하기 위한 알고리즘
ㄴ 우선순위를 바탕으로 직렬 또는 병렬로 중재하는 위의 방식들은 미리 우선순위가 세팅된 정적 우선순위 알고리즘에 해당한다.
(1) 시간 분할(time slice): 일정 시간만큼만 버스를 사용한다.
- 버스 시간을 일정한 길이의 시간으로 나누고, 라운드 로빈(round-robin) 방식으로 각 프로세스에 할당한다.
- 모든 요소들이 동등하게 버스를 사용하게 된다.
(2) 폴링(polling): 폴(poll) 라인으로 버스 승인 신호를 대체하여 프로그램으로 제어한다.
- 모든 장치에 폴(poll) 라인을 연결한다. 버스 승인 신호를 대체한다.
- 버스 제어기는 지정된 우선순위를 확인하여 버스를 할당한다. 프로그램으로 버스를 할당할 우선순위를 변경할 수 있다.
(3) LRU(Least Recently Used): 버스를 사용한 지 오래된 장치에 우선권을 준다.
- 몇 차례의 버스 사이클마다 최근 버스 사용시각 순으로 우선순위가 변경된다.
- 모든 프로세스가 버스에 접근할 기회를 동등하게 부여받을 수 있다.
(4) First-Come First-Serve: 큐(queue) 방식으로 버스 요청을 제어한다.
- 버스 요청이 도착한 순으로 버스를 할당한다.

(5) 회전 데이지 체인(Rotating Daisy-Chain): 모든 장치에 차례로 버스를 제공한다.

4) 입출력 인터럽트
- 컴퓨터는 주변 장치의 활성화를 위한 소프트웨어 루틴(I/O routine)을 가져야 한다.
 ㄴ 제어 명령 발송과 데이터 전송을 위한 준비 상태 확인 로직 필요
 ㄴ 시스템에 인터럽트 신호 전송, 입출력 실행할 주변장치에 명령 전송 기능 필요
 ㄴ 인터럽트 외에 데이지 체인같이 우선순위에 따라 직접 전송(DMA, Direct Memory Access) 하는 장치의 경우, DMA채널을 개시할 수 있어야 한다.
- CPU와 입출력장치 간 데이터 전송은 CPU에 의해 시작된다.
 ㄴ 각 장치는 제어라인의 상태나 플래그에 따라 CPU에 인터럽트를 요청해서 통신 가능
 ㄴ 입출력 인터페이스의 상태 레지스터에서 Flag bit가 켜지면, CPU는 수행 중이던 프로그램을 중지하고 I/O 전송을 수행 후 원 프로그램으로 복귀한다.
 ㄴ Vectored Interrupt: 인터럽트를 내는 소스가 CPU에게 분기에 대한 정보를 제공
 ㄴ Non-vectored Interrupt: 분기 주소가 메모리의 특정 위치에 고정되어 있는 방식
- 여러 장치가 하나의 리소스에 접근하는 등 다수의 인터럽트 발생 시 중재가 필요하다.

4-1) 우선순위 인터럽트
- 각 소스에 우선순위를 부여하는 시스템. 우선순위는 하드웨어/소프트웨어적으로 모두 처리 가능하다.
- 버스 중재 방식과 유사하다. 6-2-1) ~ 6-2-2) 참고.
 (1) 데이지 체인(직렬) 우선순위 인터럽트
 - 인터럽트를 발생하는 모든 장치들을 직렬로 연결, 순서대로 인터럽트를 요청하고 응답을 받는다.
 (2) 병렬 우선순위 인터럽트
 - 인터럽트 레지스터는 각 장치별 인터럽트 요청 여부를 각각의 비트에 저장하여 전달한다. 해당 비트는 마스크 레지스터를 통해 마스킹 후 인코더 진리표에 의해 해석되어 실행된다.

7. 병렬 컴퓨터

1) 병렬 프로세스 시스템 (Parallel Process System)
- 동시에 여러 명령 또는 여러 작업을 수행하는 병렬 처리가 가능한 시스템
 ㄴ 다중 장치 구조: 다수의 CPU로 여러 개의 작업 처리 (공간적 병렬성)
 ㄴ 파이프라인 구조: 다수의 작업을 다른 실행 단계에서 병렬로 처리 (시간적 병렬성)

* 시스템 버스와 로컬 버스
 - 시스템 버스: CPU, IOP(I/O Processor), Memory 등 주요 하드웨어의 연결 체계
 - 로컬 버스: 메모리 버스, 입출력 버스, 통신 버스 등

2) 멀티프로세서
- 메모리와 입출력장치를 공유하는 2개 이상의 CPU를 갖는 시스템

ㄴ 하나의 운영체제에 의해 시스템의 모든 요소가 제어된다. (멀티 컴퓨터와 다름)
ㄴ 하나의 작업을 여러 부분으로 나누거나, 다수의 독립적인 작업들이 병렬적으로 처리될 수 있다. 다만 프로그래밍 언어나 컴파일러 차원에서 병렬 실행 구현 기능이 제공되어야 한다.
- 연결 구조에 따라 크게 두 가지로 분류된다.
ㄴ 공유 메모리 또는 밀착결합 멀티프로세서(Shared Memory, Tightly Coupled): 프로세스와 메모리 사이의 경로 수에 따라 구성이 달라진다.
ㄴ 분리 메모리 또는 느슨히 결합된 시스템(Distributed Memory, Loosely Coupled): 프로세싱 요소들 사이의 전송 경로 수에 따라 구성이 달라진다.

2-1) 상호 연결 구조에 따른 멀티프로세서 구성 분류
- 시분할 공통 버스
(1) 단일 공통 버스 시스템: 여러 CPU가 공통의 시스템 버스 라인에 연결되어 있다.
- 충돌 발생 가능성은 제어기에 의해 해결 가능
- 시스템 내 전체 전송률이 단일 경로의 속도에 맞춰 제한될 수 있음
(2) 이중 버스 구조: CPU마다 시스템 버스 제어기를 가진 개별적인 로컬 버스를 가진다.
- 시스템 버스 제어기에 의해 로컬 버스를 시스템 버스에 연결한다.
- 각 프로세서의 대기시간이 줄어들고 동시에 여러 개의 버스 전송이 가능
- 시스템 구축비용과 복잡도가 증가하며, 시스템 버스 자체는 여전히 주어진 시간에 하나의 프로세서만 사용할 수 있다.
- 다중 포트 메모리
(1) 다중 포트 메모리: 프로세스 버스(CPU)와 메모리 모듈이 각각 버스라인으로 연결된다.
- 각 메모리 모듈은 어떤 포트가 메모리에 접근할 수 있는가를 결정할 논리 회로를 가진다.
ㄴ 일반적으로 우선순위는 포트의 위치에 따라 결정됨
- 프로세스와 메모리 사이의 높은 전송률을 보인다.
- 고가의 메모리 제어 논리회로와 전선 커넥터가 필요하다.
(2) 크로스바 스위치: 프로세스 버스와 메모리 모듈의 통로 사이에 크로스포인트가 있다.
- 크로스포인트(crosspoints): 프로세서와 메모리 모듈 간의 통로를 결정한다.
ㄴ 버스에 실린 주소를 확인하여 통로를 열어주거나, 우선순위에 따라 중재
(3) 다단 교환망: 2입력, 2출력 상호교환 스위치를 사용한다.
- 크로스포인트와 유사한 구조지만, 크로스포인트와 달리 2개의 입력을 받는다.
- 2개의 입력 중 하나를 선택하여 전체 경로를 연결하며, 이를 통해 충돌을 중재한다.
ㄴ 입력-출력을 연결할 제어 신호가 필요하다.
(4) 하이퍼큐브 상호연결: 2^n개의 프로세서가 n차원 이진 큐브로 연결된 입체적인 형태
- 느슨히 결합된 시스템의 일종으로, 각 프로세서는 큐브의 노드를 형성한다.
ㄴ 노드에는 CPU, 로컬 메모리, 입출력 인터페이스도 포함된다.

3) 프로세서간 통신과 동기화
- 여러 프로세서들은 공통의 입출력 채널을 통해 통신한다.
- 공통 메모리: 모든 프로세서가 접근할 수 있도록 할당한 공간
ㄴ 프로세서들 사이에서 오가는 메시지를 저장 및 송/수신하는 역할을 한다.
- 멀티프로세서를 위한 운영체제 종류

(1) 주종 모드(master-slave)
- 주 프로세서(master)가 OS기능 수행, 종 프로세서(slave)는 인터럽트 요청
- 밀착결합 형태에 적합하다.
(2) 분리 운영체제(Separate Operating System)
- 모든 프로세서가 자신의 운영체제를 가진다.
- 느슨한 결합 형태에 적합하다.
(3) 분산 운영체제(Distributed Operating System)
- 운영체제가 여러 프로세서에 분산되어 있다.
- OS의 특정 기능은 한 번에 하나의 프로세서에서만 작동된다.
- 프로세서간 동기화: 공용 변수 기록 시 상호배제(Mutual Exclusive)적 접근 필요
ㄴ 하드웨어적으로 이진 세마포어(Semaphore)를 이용하는 기법이 많이 활용된다.
- 캐시 일관성: 프로세서별 로컬 메모리(캐시)의 내용이 동일하게 유지되어야 한다는 정책
ㄴ 프로세서마다 캐시가 있는데, 공용 변수를 캐시에 저장하고 변경하면 다른 캐시와 동일한 값을 보장할 수 없다.
ㄴ 주기억장치에 공용 캐시를 두는 방식 -> 근접성 원리 위배, 접근 시간이 증가하는 문제
ㄴ 읽기 전용 데이터만 캐싱 -> 일부 데이터만 캐싱 가능, 데이터 종류를 선별하는 번거로움
ㄴ 스누피 캐시 제어기(snoopy cache controller): 버스에 부착된 모든 캐시를 감시, 한 캐시에 변형이 일어나는 순간 모든 캐시를 무력화시키는 하드웨어 장치 -> 복잡도 증가