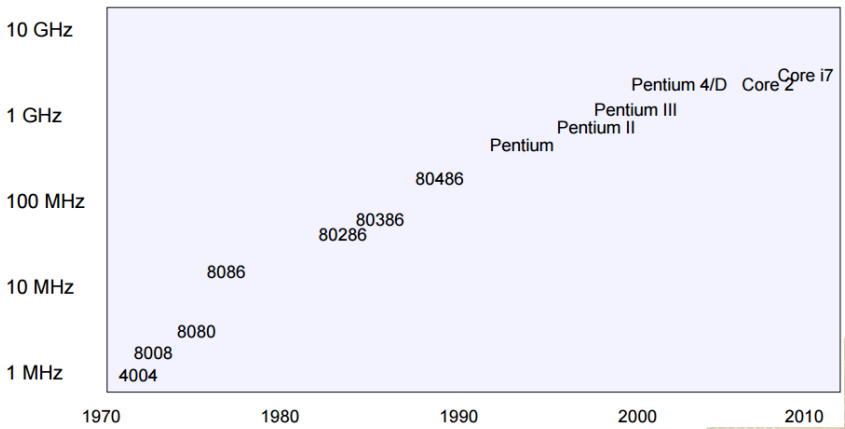
# 제12장
# x86 프로세서



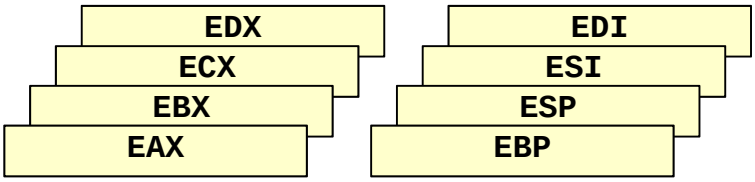

## x86 프로세서 개발 역사

- 1970년 4비트 프로세서 4004에서 시작
- 1981년 IBM PC에 적용되면서 급속히 확산

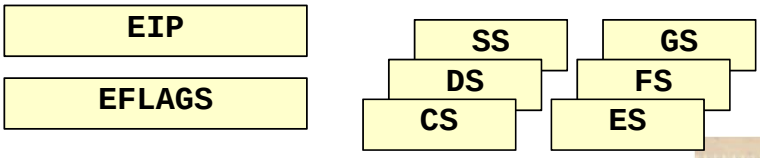


# x86 프로세서의 레지스터 집합

32비트 범용 레지스터



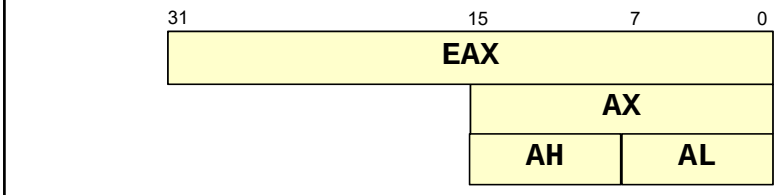

16비트 세그먼트 레지스터

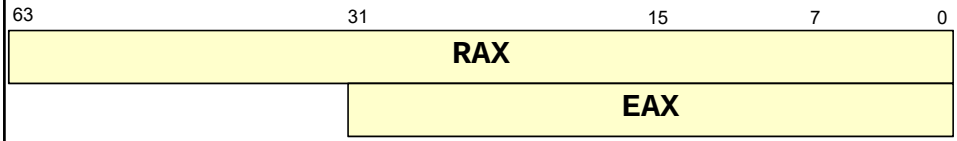







# 범용 레지스터의 일부분 지정



## 기타 레지스터 들

- Floating point registers: ST0 ~ ST7
- MMX registers, SSE registers, …
  - ✓MMX: Multimedia Extensions
  - ✓SSE: Streaming SIMD Extensions
- Control Registers
- …





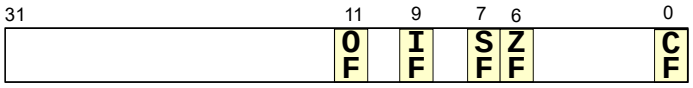
## 명령어 구조

- 명령어당 1 ~ 15 byte
- Prefix 0 ~ 4 byte
- Opcode 1 ~ 2 byte, 명령어에 따라서 ModReg의 일부도 사용
- ModReg 1 byte
- IndBase 1 byte (Scale, Index, Base)
- Displacement 0 ~ 4 byte
- Immediate 0 ~ 4 byte

| Prefix | Opcode | ModReg | IndBase | Disp. | Imm. |
|---|---|---|---|---|---|

# EFLAGS 레지스터

## EFLAGS 레지스터

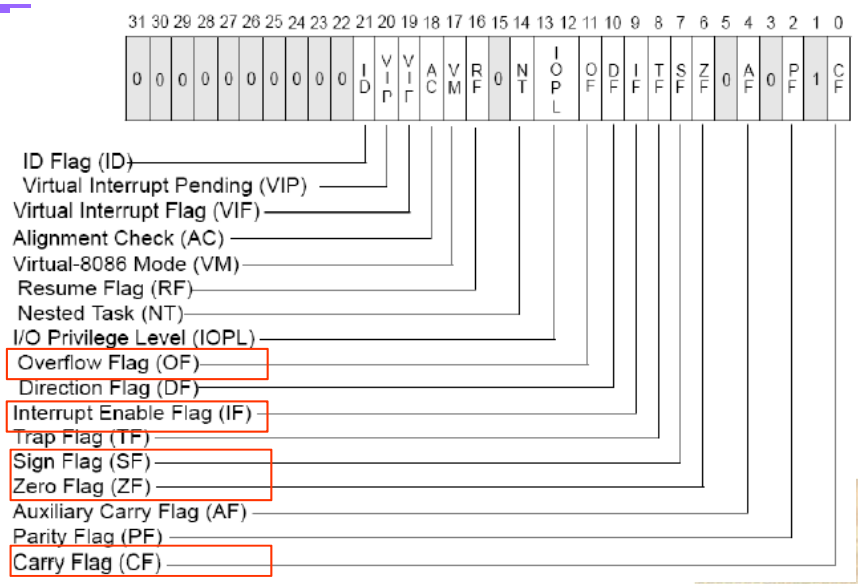



OF: Overflow Flag
IF: Interrupt Enable Flag
SF: Sign Flag
ZF: Zero Flag
CF: Carry Flag





# (참고) EFLAGS 레지스터



## 명령어 집합

- 2 오퍼란드 형식
  - ✓ addl %ebx, %eax ; EAX ← EAX + EBX
- 스택에 대한 push, pop 명령어 제공
  - ✓ pushl %ebp
  - ✓ popl %eax
- 다양한 명령어들에 메모리 접근 기능 포함
  - ✓ negl variable
  - ✓ addl 4(%edi), %edx
  - ✓ addl $3, 0x2000
- MUL, DIV, Floating point 연산, …
- 총 300여 개의 명령어 (Pentium 기준)





## 명령어 집합 (Pentium 기준)

- 총 300여 개의 명령어
- 정수 산술논리 연산: (SZCO 비트 갱신)
  - ✓ ADD, ADC, SUB, CMP, MUL, DIV, …
- Data Transfer:
  - ✓ MOV, XCHG, PUSH, POP, IN, OUT, …
- 분기 관련:
  - ✓ JMP, JE, JGE, LOOP, …
- 함수호출 / 소프트웨어 인터럽트:
  - ✓ CALL, RET, INT, IRET, …
- 기타
  - ✓ Floating Point, 문자열 처리, EFLAGS 관련, MMX, …

# TOY와 x86의 명령어 비교 (1/2)

| TOY | x86 | 비고 |
|---|---|---|
| ADD R0, R0, R1<br>ADD R0, R0, 7 | addl %ebx, %eax<br>addl $7, %eax | EAX ← EAX + EBX<br>EAX ← EAX + 7 |
| SUB R0, R0, R1 | subl %ebx, %eax | EAX ← EAX - EBX |
| CMP R0, 7 | cmpl $7, %eax | EAX - 7 |
| AND R1, R1, R2 | andl %ecx, %ebx | EBX ← EBX and ECX |
| OR R1, R1, 7 | orl $7, %ebx | EBX ← EBX or 7 |
| COPY R0, R1 | movl %ebx, %eax | EAX ← EBX |
| COPY R0, 7 | movl $7, %eax | EAX ← 7 |
| NOT R0, R0 | notl %eax | EAX ← not EAX |
| XOR R0, R0, R1 | xorl %ebx, %eax | EAX ← EAX xor EBX |
| LSL R0, R0, 1 | shll %eax | |
| LSR R0, R0, 1 | shrl %eax | |
| ASL R0, R0, 1 | sall %eax | EAX ← EAX << 1 |
| ASR R0, R0, 1 | sarl %eax | EAX ← EAX >> 1 |





# TOY와 x86의 명령어 비교 (2/2)

| TOY | x86 | 비고 |
|---|---|---|
| LOAD R0, 0x2000 | movl 0x2000, %eax | EAX ← M[0x2000] |
| LDR R0, R1, 4 | movl 4(%ebx), %eax | EAX ← M[EBX+4] |
| STORE R0, 0x2000 | movl %eax, 0x2000 | M[0x2000] ← EAX |
| STR R0, R1, 4 | movl %eax, 4(%ebx) | M[EBX+4] ← EAX |
| BR nzp, 0x2000<br>BR z, 0x2000 | jmp 0x2000<br>jeq 0x2000 | |
| BRR nzp, R0, 0 | jmp %eax | |
| LEA R0, 0x2000 | LEA 0x2000, %eax | |
| LINK<br>BR nzp, 0x2000 | call 0x2000 | 함수 호출 |
| RET | ret | 함수에서 복귀 |
| SWI 5 | int $5 | 운영체제 기능 호출 |
| RTI | iret | 운영체제에서 복귀 |
| SUB R5, R5, 1<br>STR R0, R5, 0 | pushl %eax | 스택 PUSH<br>(한 명령어로 처리) |
| LDR R0, R5, 0<br>ADD R5, R5, 1 | popl %eax | 스택 POP<br>(한 명령어로 처리) |

## x86의 최대공약수 구하기 프로그램

```
...
gcd: cmpl %ebx, %eax      ; EAX을 EBX와 비교
     je stop              ; 일치하면 stop로 분기
     jl less              ; 작으면 less로 분기
     subl %ebx, %eax      ; EAX ← EAX - EBX
     jmp gcd              ; 무조건 gcd로 분기
less: subl %eax, %ebx     ; EBX ← EBX - EAX
     jmp gcd              ; 무조건 gcd로 분기
stop: ...
...
```





## x86의 최대공약수 구하기 프로그램

TOY

```
     ...
gcd:CMP R0, R1
     BR z, stop
     BR n, less
     SUB R0, R0, R1
     BR nzp, gcd
less: SUB R1, R1, R0
     BR nzp, gcd
stop: ...
...
```

x86

```
...
gcd: cmpl %ebx, %eax  ; EAX을 EBX와 비교
       je stop        ; 일치하면 stop로 분기
       jl less        ; 작으면 less로 분기
       subl %ebx, %eax ; EAX ← EAX - EBX
       jmp gcd        ; 무조건 gcd로 분기
less: subl %eax, %ebx ; EBX ← EBX - EAX
       jmp gcd        ; 무조건 gcd로 분기
stop: ...
...
```

## 함수 호출

- 함수 호출
  - ✓CALL func
  - ✓동작:
    - **스택에 EIP Push**
    - **EIP ← func**
- 함수에서 복귀
  - ✓RET
  - ✓동작:
    - **스택에서 EIP Pop**
- 컴파일러
  - ✓함수의 인수들도 스택에 Push하는 것을 전제로 함
  - ✓함수실행에서 스택 프레임 기준점은 EBP 사용
  - ✓함수 시작부분: EBP를 스택에 Push후에 ESP를 EBP에 복사
  - ✓함수 끝부분: 스택에서 Pop 하여 이전 EBP 값을 복구





## x86의 함수 표현 예

- C 언어 프로그램

```
int sum(int a, int b)
{     int result;

      result = a + b;
      return result;
}
```

- 어셈블리 언어 표현

```
sum:   pushl %ebp
       movl %esp, %ebp
       subl $4, %esp
       movl 8(%ebp), %eax
       addl 12(%ebp), %eax
       movl %eax, -4(%ebp)
       movl -4(%ebp), %eax
       addl $4, %esp
       popl %ebp
       ret
```

## x86의 함수 호출 예

- C 언어 프로그램

```
int x, y;
main()
{
    …
    y = sum(x, 5);
    …
}
```

- 어셈블리 언어 표현

```
…
pushl $5
pushl x;
call sum
addl $8, %esp
movl %eax, y
…
```





## 운영체제 기능 호출

- 운영체제 기능 호출
  - ✓INT n
  - ✓동작:
    - **스택에 EFLASG Push**
    - **스택에 EIP Push**
    - **EFLASG ← 운영체제 모드, EIP ← M[n*4]**
- 운영체제에서 복귀
  - ✓IRET
  - ✓동작:
    - **스택에서 EIP Pop**
    - **스택에서 EFLASG Pop**