2.4 명령어 세트(instruction set)

연세대학교

- 명령어 세트
  - CPU가 지원하는 기계어 명령어들의 집합
- 명령어 연산의 종류
  - 데이터 전송 : 레지스터/메모리 간에 데이터 이동
  - 산술 연산 : 덧셈, 뺄셈, 곱셈 및 나눗셈
  - 논리 연산 : 비트들 간의 AND, OR, NOT 및 XOR 연산
  - 입출력(I/O) : CPU(레지스터)와 외부 장치들 간의 데이터 이동
  - 프로그램 제어: 분기, 서브루틴 호출(subroutine call)
    - 명령어 실행 순서를 변경하는 연산들
    - 무조건 분기, 조건부 분기

오퍼랜드의 수에 따른 명령어 분류

연세대학교

- 1-주소 명령어(one-address instruction) :

  [예] ADD X ; AC ← AC + M[X]
  - AC 레지스터가 묵시적 오퍼랜드로 사용됨
- 2-주소 명령어(two-address instruction) :

  [예] ADD R1, R2 ; R1 ← R1 + R2

  MOV R1, R2 ; R1 ← R2

  ADD R1, X ; R1 ← R1 + M[X]
  - 한 오퍼랜드는 source와 destination으로 동시에 함
  - 오퍼랜드는 레지스터 또는 메모리이며, 동시에 메모리인 것을 허용하지 않는 CPU들도 많이 있음
- 3-주소 명령어(three-address instruction) :

  [예] ADD R1, R2, R3 ; R1 ← R2 + R3
  - 대부분의 3주소 명령어의 오퍼랜드는 레지스터만 가능함

범용 레지스터

연세대학교

- Accumulator 방식의 CPU
  - 모든 연산 결과는 AC(Accumulator)에 저장함

  (예) 앞에서 소개한 간단한 CPU, 1주소 명령어
- 범용 레지스터를 사용하는 CPU
  - 대개의 CPU는 연산결과를 저장할 수 있는 여러 개의 범용 레지스터를 제공함

  (예) **ARM** : R0-R15

  80386 : EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP

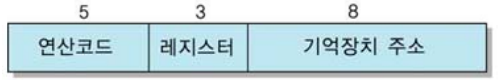
  64-bit x86: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, R8-R15
  - 명령어 형식에서 레지스터 필드에 사용하는 레지스터를 지정함

여러 명령어 형식

연세대학교

- 1-주소 명령어

- 2-주소 명령어



- 3-주소 명령어

- 명령어의 길이는 (1) 고정 길이 형식 (2) 가변 길이 형식

명령어 형식

연세대학교

- 연산 코드(Operation Code) :
  - 수행될 연산을 지정
  - 연산 코드 필드 길이 : 연산의 개수를 결정
    - (예) 4 비트 → 2^4 = 16 가지의 연산 정의 가능
    - 연산 코드 필드가 5 비트로 늘어나면, 2^5 = 32 가지 연산들 정의 가능하지만 다른 필드의 길이가 감소됨
- 오퍼랜드(Operand)
  - 연산을 수행하는 데 필요한 데이터 또는 데이터의 위치를 지정
    - 데이터는 CPU 레지스터, 주기억장치, 또는 I/O 장치에 위치
      레지스터 번호 메모리주소
  - 분기/호출 명령어에서는 분기할 명령어 주소를 포함

연세대학교

- 오퍼랜드 필드의 길이 : 오퍼랜드의 범위를 결정
  - 오퍼랜드의 종류에 따라 범위가 달라짐
    - 데이터 : 표현 가능한 수의 크기가 결정
    - 메모리 주소 : CPU가 직접 주소를 지정할 수 있는 메모리 크기를 결정
    - 레지스터 번호 : 데이터 저장에 사용될 수 있는 레지스터의 수를 결정

명령어 형식이 프로그래밍에 미치는 영향

연세대학교

[예] X = (A + B) x (C - D)

- 프로그래밍에 다음과 같은 니모닉을 가진 명령어들을 사용
  - ADD : 덧셈
  - SUB : 뺄셈
  - MUL : 곱셈
  - DIV : 나눗셈

  - MOV : 데이터 이동
  - LOAD : 기억장치로부터 레지스터로 데이터 적재
  - STOR : 레지스터에서 기억장치로 데이터 저장

1-주소 명령어를 사용한 프로그램

연세대학교

- 프로그램

```
LOAD   A      ; AC ← M[A]
ADD    B      ; AC ← AC + M[B]
STOR   T      ; M[T] ← AC          T ← A+B
LOAD   C      ; AC ← M[C]
SUB    D      ; AC ← AC - M[D]     AC ← C - D
MUL    T      ; AC ← AC * M[T]     AC ← (A+B)*(C-D)
STOR   X      ; M[X] ← AC          X ← AC
```

  - M[A] : 메모리 A번지의 내용
  - T : 메모리 내의 임시 저장장소의 주소
    (레지스터가 AC 한 개 밖에 없으므로 임시 저장장소를 사용)

- 프로그램의 길이 = 7 명령어

2-주소 명령어를 사용한 프로그램

연세대학교

- 프로그램

```
MOV     R1, A     ; R1 ← M[A]
ADD     R1, B     ; R1 ← R1 + M[B]      R1 ← A+B
MOV     R2, C     ; R2 ← M[C]
SUB     R2, D     ; R2 ← R2 - M[D]      R2 ← C-D
MUL     R1, R2    ; R1 ← R1* R2
MOV     X, R1     ; M[X] ← R1
```

- 프로그램의 길이 = 6 명령어

3-주소 명령어를 사용한 프로그램

연세대학교

- 프로그램

```
ADD   R1, A, B        ;  R1 ← M[A] + M[B]
SUB    R2, C, D       ;  R2 ← M[C] - M[D]
MUL   X, R1, R2       ;  M[X] ← R1 × R2
```

- 프로그램의 길이 = 3
- 단점
  - 명령어의 길이가 증가한다 (오퍼랜드가 3개)
  - 명령어 해독 과정이 복잡해진다
- **RISC(Reduced Instruction Set Computer)** 형식의 명령어는 레지스터 오퍼랜드만 사용하는 3 주소 명령어를 제공

CISC와 RISC 명령어 집합

연세대학교

- CISC – complex instruction set computer
  - 많은 수의 명령어들로 구성됨
  - 복잡한 동작의 고수준 연산을 수행하는 명령어들이 포함됨
    - 메모리 오퍼랜드에 대해서 산술/논리 연산 수행 허용
  - 마이크로코드(micro-code) 에 의한 기계어 해독

  (예) Intel 80x86 계열, 680x0계열
- RISC – reduced instruction set computer
  - 적은 수의 짧고 간단한 명령어들로 구성됨
  - 하드웨어에 의한 기계어 처리
  - 레지스터 오퍼랜드만 사용하는 3-주소 연산 지원
  - load와 store 명령어를 사용하여 메모리 오퍼랜드를 액세스
  - 프로그램의 길이가 길어짐 (명령어를 더 많이 사용)

  (예) Sparc, PowerPC, ARM, Alpha ...

RISC의 3-주소 명령어 프로그램

연세대학교

- 프로그램

```
LOAD R1, A
LOAD R2, B
ADD   R1, R1, R2      ;  R1 ← M[A] + M[B]
LOAD R2, C
LOAD R3, D
SUB    R2, R2, R3     ;  R2 ← M[C] - M[D]
MUL   R1, R1, R2
STORE  X, R1          ;  M[X] ← R1*R2
```

- 프로그램 길이 = 8 명령어
- RISC는 프로그램 길이가 길어지지만 각 명령어의 실행시간이 비슷하여 명령어 파이프라이닝의 효율이 좋음

상태 레지스터(status register)

연세대학교

- 상태 레지스터
  - CPU 연산의 실행 결과에 대한 상태(조건 플래그)를 저장
  - 조건분기 명령어에서 사용: 조건 플래그에 따라서 분기여부 결정
- 플래그
  - 부호(Sign, Negative) 플래그 : 산술연산 결과의 부호 비트를 저장
  - 영(Zero) 플래그 : 연산 결과 값이 0 이면 1로 설정
  - 올림수(Carry) 플래그 : 덧셈이나 뺄셈에서 올림수(carry)나 빌림수(borrow)가 발생한 경우에 1로 설정
  - 오버플로우(oVerflow) 플래그 : 산술 연산 과정에서 오버플로우가 발생한 경우에 1로 설정

약칭: **C, V, N, Z** 또는 **CF, OF, SF, ZF**

조건 분기 명령어

연세대학교

- 형식

  Jcc addr (cc는 조건을 명시함)
  - 이전 명령어의 실행 결과가 cc로 지정되는 조건을 만족하면 addr 번지로 분기하며, 그렇지 않으면 다음 명령어를 수행함.
- 플래그 값에 따른 분기

JS addr	JNS addr	(Sign)
JC addr	JNC addr	(Carry)
JZ addr	JNZ addr	(Zero)
JO addr	JNO addr	(Overflow)

- 크기 비교에 따른 분기 - 조건 플래그들을 참조하여 결정

JE addr	JNE addr			
JG addr	JGE addr	JL addr	JLE addr	(부호있는 비교)
JA addr	JAE addr	JB addr	JBE addr	(부호없는 비교)

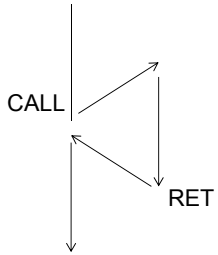
  - 부호있는 비교와 부호없는 비교를 구분하여 조건분기명령어 제공

서브루틴 호출을 위한 명령어

연세대학교

- CALL 명령어 : 서브루틴을 호출함
  - 동작:

    1) 현재의 PC 내용을 스택에 저장하고
    2) 서브루틴의 시작 주소로 분기



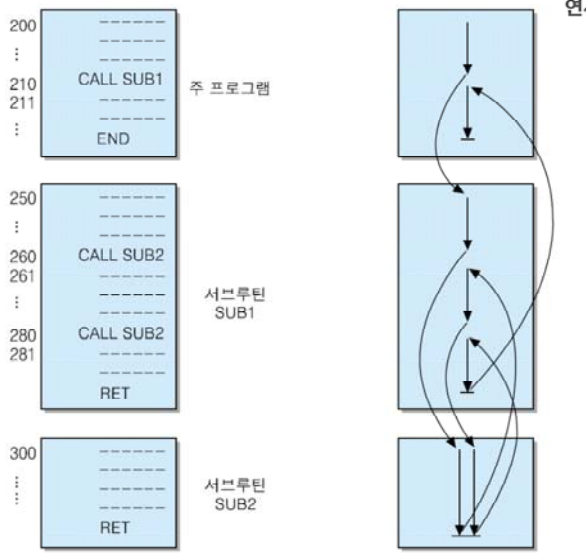
- RET 명령어 : 원래 실행하던 루틴으로 복귀
  - 동작:

    1) 스택에 저장된 복귀주소로 분기

- 스택(stack)
  - Last In First Out 동작을 하는 자료구조
  - nested 호출시에 복귀주소가 사용되는 순서가 저장되는 순서와 반대이므로 스택의 동작과 같음

서브루틴이 포함된 프로그램이 수행되는 순서

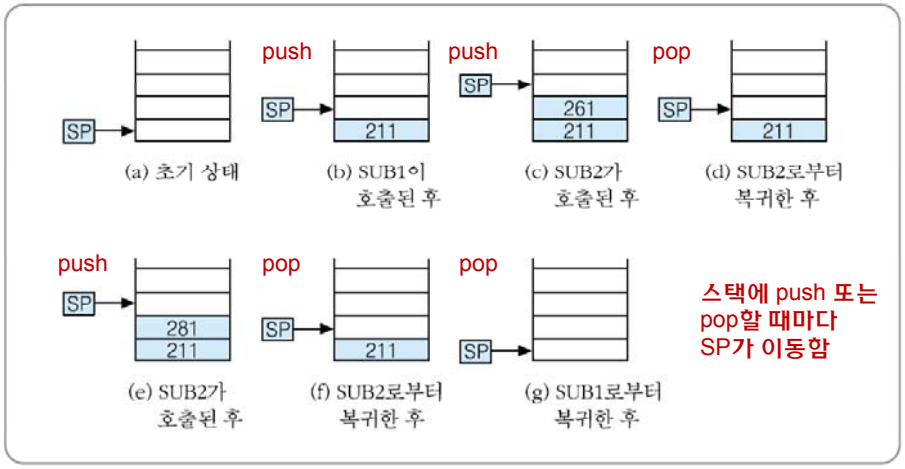
연세대학교



프로그램 수행 과정에서 스택의 변화

연세대학교

- SP(Stack Pointer) : 메모리의 스택 영역의 꼭대기 주소를 가리킴



주소지정 방식

연세대학교

- 주소지정방식(addressing mode)
  - 오퍼랜드의 위치를 지정하는 방식
- 다양한 주소지정 방식(addressing mode)을 사용하는 이유 :
  - 제한된 수의 명령어 비트들을 이용하여 여러 가지 방법으로 오퍼랜드를 지정하고, 고급언어에서 간편하게 기계어 명령어를 생성할 수 있도록 하기 위함
- 기호
  - EA : 유효 주소(Effective Address),
    - 데이터가 저장된 기억장치의 실제 주소
  - A : 명령어 내의 주소 필드 내용
  - M(A) : 기억장치 A 번지의 내용
  - R필드 : 명령어 내의 레지스터 필드 내용
  - R : R필드가 가리키는 레지스터의 내용

주소지정 방식의 종류

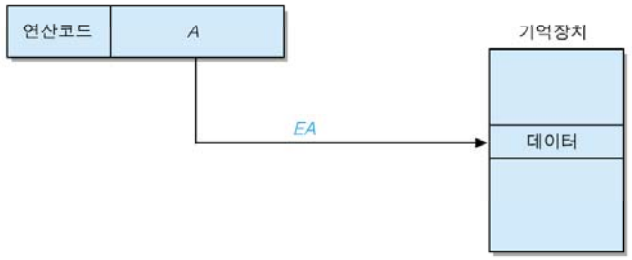
연세대학교

- 직접 주소지정 방식 (direct addressing mode)
- 간접 주소지정 방식 (indirect addressing mode)
- 묵시적 주소지정 방식 (implied addressing mode)
- 즉치 주소지정 방식 (immediate addressing mode)
- 레지스터 주소지정 방식 (register addressing mode)
- 레지스터 간접 주소지정 방식 (register-indirect addressing mode)
- 변위 주소지정 방식 (displacement addressing mode)
  - 상대 주소지정 방식(relative addressing mode)
  - 인덱스 주소지정 방식(indexed addressing mode)
  - 베이스-레지스터 주소지정 방식(base-register addressing mode)

직접 주소지정 방식

연세대학교

- 유효주소 EA = A
- 오퍼랜드 = M(A)



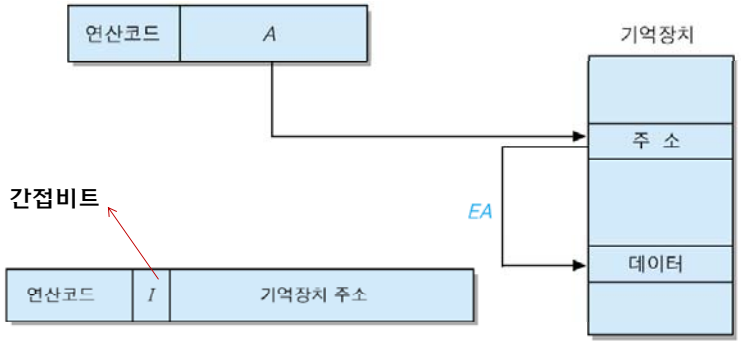
MOV R1, 100 ; R1 ← M(100)

간접 주소지정 방식

연세대학교

- 유효주소 EA = M(A)
- 오퍼랜드 = M(EA) = M(M(A))

MOV R1, (100) ; R1 ← M(M(100))



묵시적 / 즉치 주소지정 방식

연세대학교

- 묵시적 주소지정 방식: 명령어 실행에 필요한 데이터의 위치가 묵시적으로 지정되는 방식
  - 1-주소 명령어 : 누산기가 오퍼랜드로 사용됨
  - PUSH R1 명령어 : 레지스터 R1의 내용을 스택에 저장
    EA = 미리 정해진 레지스터/기억장소
    오퍼랜드 = 미리 정해진 레지스터/메모리의 내용
- 즉치 주소지정 방식: 데이터가 명령어에 포함되어 있는 방식
  - 오퍼랜드 = 명령어의 데이터 필드

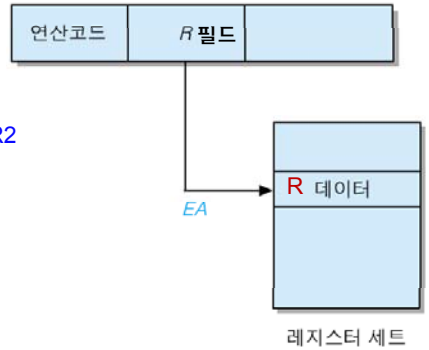
MOV R1, #100 ; R1 ← 100

레지스터 주소지정 방식

연세대학교

- 유효주소 EA = R필드
- 오퍼랜드 = R
- 주소지정에 사용될 수 있는 레지스터들의 수 = 2^k 개
  (k는 오퍼랜드 비트 수)

MOV R1, R2 ; R1 ← R2

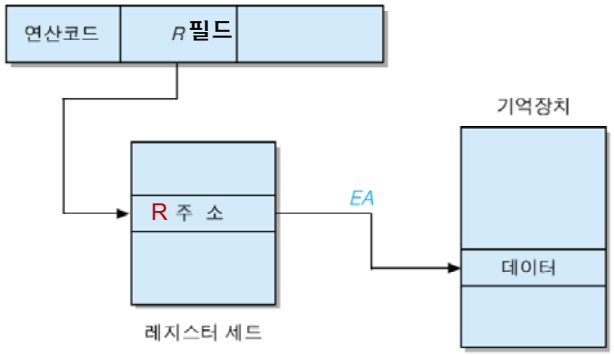


레지스터 간접 주소지정 방식

연세대학교

- 유효주소 EA = R
- 오퍼랜드 = M(R)

MOV R1, (R2) ; R1 ← M(R2)



변위 주소지정 방식

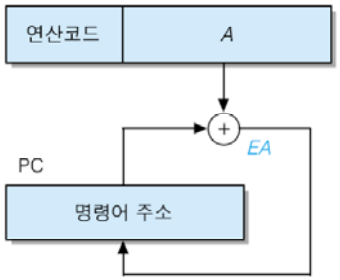
연세대학교

- 유효주소
  - EA = R + A (R:베이스, A: 변위, 양수/음수가능)
  - EA = A + R (A:시작주소, R: 인덱스)
- 사용되는 레지스터에 따라 여러 종류의 변위 주소지정 방식 정의
  - 상대 주소지정 방식: R = PC, EA = **PC** + A
  - 인덱스 주소지정 방식: R = 인덱스 레지스터, EA = A + **IX**
  - 베이스레지스터 주소지정 방식: R = 베이스레지스터, EA = **BR**+A

A는 상수, R(레지스터)는 변수

상대 주소지정 방식

연세대학교



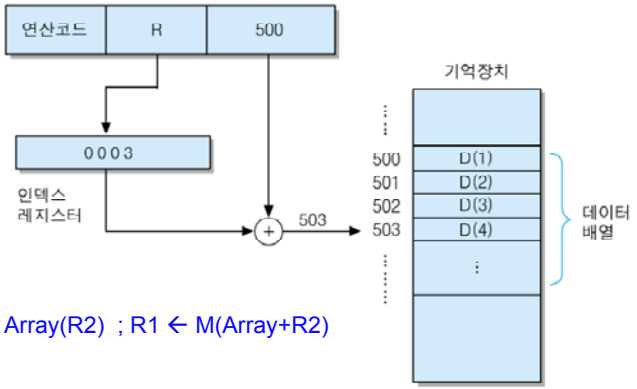
JMP target
→ target 분기주소가 상대주소로 표기

MOV R1, Var
→ Var에 대한 A 필드가 상대주소로 표기

- 장점 :
  - 전체 주소 비트수 보다 적은 수의 비트로 분기 주소를 표현가능
  - 프로그램 위치가 이동해도 오퍼랜드의 상대주소는 불변
- 단점 :
  - 분기 범위가 오퍼랜드 필드의 길이에 의하여 제한

인덱스 주소지정 방식

연세대학교



MOV R1, Array(R2) ; R1 ← M(Array+R2)

- 주요 용도 : 배열 데이터 액세스
  - 배열의 첨자 값을 인덱스 레지스터에 저장하여 사용

자동 증가, 감소 주소지정방식

연세대학교

- 자동 인덱싱(auto-indexing)
  - 명령어가 실행될 때마다 (인덱스) 레지스터의 내용이 자동적으로 증가 혹은 감소됨
    - 사전증감: 명령어가 실행되기 전에 증가 혹은 감소
    - 사후증감: 명령어가 실행된 후에 증가 혹은 감소
  - 용도: 스택연산, 블록 데이터 이동

MOV R1, Array(R2+) ; R1 ← M(Array+R2), R2←R2+1

MOV R1, Array(-R2) ; R2←R2-1, R1 ← M(Array+R2)

베이스-레지스터 주소지정 방식

연세대학교

- 유효주소 EA = BR + A
  - 베이스 레지스터의 내용과 변위 A를 더하여 유효 주소를 결정
- 주요 용도 :
  - 서로 다른 세그먼트 내 프로그램의 위치 지정
  - 구조체의 특정 필드 액세스

MOV R1, (R2+10) ; R1 ← M(R2+10)

펜티엄 프로세서의 명령어 형식

연세대학교

- 선형 주소(linear address: LA) = 세그먼트의 시작 주소 + 유효주소
  (세그먼트의 시작 주소는 세그먼트 레지스터에 저장)

주소지정 방식	유효 주소(*EA*)	선형 주소(*LA*)
즉치 방식	데이터 = *A*	
레지스터 방식	*EA* = R	*LA* = *R*
변위 방식	*EA* = *A*	*LA* = (SR) + *EA*
베이스 방식	*EA* = (BR)	*LA* = (SR) + *EA*
변위를 가진 베이스 방식	*EA* = (BR) + *A*	*LA* = (SR) + *EA*
변위를 가진 인덱스 방식	*EA* = (IX) + *A*	*LA* = (SR) + *EA*
인덱스와 변위를 가진 베이스 방식	*EA* = (IX) + (BR) + *A*	*LA* = (SR) + *EA*
상대 방식	*EA* = (PC) + *A*	*LA* = *EA*

Pentium의 가변 길이 명령어 형식

연세대학교

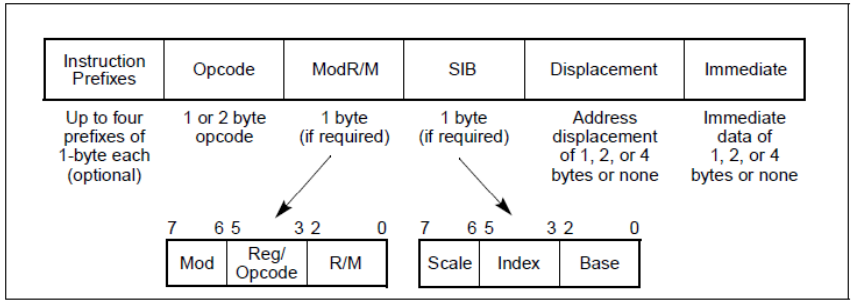


Figure 2-1. Intel Architecture Instruction Format

ARM Instruction format

연세대학교

3 1	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	
Cond			0	0	I	Opcode				S	Rn			Rd				Operand 2												*Data Processing / PSR Transfer*
Cond			0	0	0	0	0	0	A	S	Rd			Rn				Rs			1	0	0	1	Rm					*Multiply*
Cond			0	0	0	0	1	U	A	S	RdHi			RdLo				Rn			1	0	0	1	Rm					*Multiply Long*
Cond			0	0	0	1	0	B	0	0	Rn			Rd				0	0	0	0	1	0	0	1	Rm				*Single Data Swap*
Cond			0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	Rn				*Branch and Exchange*
Cond			0	0	0	P	U	0	W	L	Rn			Rd				0	0	0	0	1	S	H	1	Rm				*Halfword Data Transfer: register offset*
Cond			0	0	0	P	U	1	W	L	Rn			Rd				Offset			1	S	H	1	Offset					*Halfword Data Transfer: immediate offset*
Cond			0	1	I	P	U	B	W	L	Rn			Rd				Offset												*Single Data Transfer*
Cond			0	1	1																				1					*Undefined*
Cond			1	0	0	P	U	S	W	L	Rn			Register List																	*Block Data Transfer*
Cond			1	0	1	L	Offset																								*Branch*
Cond			1	1	0	P	U	N	W	L	Rn			CRd				CP#			Offset										*Coprocessor Data Transfer*
Cond			1	1	1	0	CP Opc				CRn			CRd				CP#			CP			0	CRm					*Coprocessor Data Operation*
Cond			1	1	1	0	CP Opc			L	CRn			Rd				CP#			CP			1	CRm					*Coprocessor Register Transfer*
Cond			1	1	1	1	Ignored by processor																								*Software Interrupt*