|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **이미지** | **해설** | | |  | - 트랜지스터 - | | |  | 베이스 P 또는 N 핀에 전류가 흐르게 되면  양쪽에있는 n n 또는 p p를 연결시킨다  **트랜지스터는 전류의 흐름을 제어한다** | | |  | IC( Integrated Circuit ) 통합 회로  집접회로  트랜지스터 ,저항 ,콘덴서 ,다이오드 등을  하나의 칩에 설계하여 만듬.  더이상 트랜지스터를 이용하여  직접 회로게이트를 만들 필요가 없어짐 | | |  | IC칩의 종류에 따라  And연산게이트 IC칩  Or연산 게이트 IC칩  등등 각목적에 맞는 IC칩의 등장 | | |  | ALU( Arithmetic and Logical Unit )  산술 논리 연산 장치  각각의 역활을 하는 IC칩들을 ALU안에 전부 구현.  레지스터에서 입력을 받아 모든 IC(회로)계산을 하고  opcode중 멀티플렉서 를 이용해 원하는 연산을 하고 레지스터에 연산된 값을 반환 | | |  | Controller Unit ( 제어장치 )  시스템 버스의 신호제어 Ram의 oper를 읽어와라 등등 이런 신호를 Controller Unit이 전송  IR ( Instruction Registe ) 명령 레지스터가  명령어를 해석(바이너리코드로) 하고  명령 레지스터 에 저장한다  ( 램의 명령어를 해석 하고 읽어온다명령어를 실행)  ALU의 입력 오퍼랜드에 입력값을 전달하는 역활도함  IR( Instruction Register )  명령 레지스터가  PC에서 실행한 오퍼와 오퍼랜드를 바이너리 코드로  번역하기 위해 잠시 보관 해두는 저장소 이고  번역이 된 코드는 ALU의 입력오퍼랜드로 전달한다  PC( Program Counter )  프로그램 카운터  다음번 실행할 오퍼의 주소값 을 저장 한다. | | |  | |  | CPU의 폰노이만 구조의 명령사이클  CPU가 명령어 ,기계어 하나를 실행하는 과정  [ Memory ] - Fetch( 인 출 )  [ ControllerUnit ] - Decode  Opcode,Operend 해석하는 단계  [ ALU ] - Execute  연산 ,실행 단계  [ Memory ] - WriteBack  레지스터 또는 메모리에 저장하는 행위 | | |  | ISA( Instruction Set Architecture )  명령 집합 구조  각기다른 CPU구조 에서 해당 CPU의 구조에  맞게 실행될 수 있게 적절히 ISA규칙에 맞는  기계어로 번역된다  즉 ISA는 CPU가 어떤구조의 기계어를 이해할 수 있는지 판단하여 코드를 컴파일 하게되면 해당 CPU에 맞는 기계어로 번역된다. | | | 6502 명령어 테이블  각 CPU에 해당하는 명령어 테이블이 존재 | | | | 바이너리 코드를 명령테이블에 따라 CPU의 ISA규칙에맞는 어셈블리언어로 변환한 과정 ( 디코딩 ) | | | |  | |  | |  | | 6502 CPU  레지스터 6개 ( 각 8비트 )  8비트 cpu  명령어( opcode ) 수 56개 | |  | | 6502 CPU 사양 | |  | | 6502 핀 데이터시트  데이터 핀 : D0 ~ D7 ( 8비트 )  주소 핀 : A0 ~ A15 ( 16비트 )  전원 +극 : VDD  접지( Ground ) -극 : VSS  RDY ( Ready ) Cpu사용 준비가 되었다는 걸 의미  준비가 되었다면 항상 1( High )로 연결  IRQB ( Interrupt Request )  외부장치의 인터럽트 요청을 수행한다.  NMIB ( Non - Maskable Interrupt )  하드웨어의 비정상적인 종료같은 예외상황에 대한 인터럽트 요청을 수행한다.  RESB ( Reset )  CPU의 재부팅(리셋)을 한다.  재부팅시 프로그램의 시작 주소(리셋벡터)인  fffc 와 fffd 의 주소를 읽어서 jump하여 실행한다  SOB ( Set Overflow )  연산 결과가 해당 메모리 비트를 초과하여  오버플로우가 발생하면 프로그램 을 멈추게 할지 설정한다 [ 오버플로우 플래그 설정 ] 이라고 한다  PHI2 ( Phase 2 In )  클럭의 입력 신호를 받아서  cpu의 명령 사이클을 실행한다  BE ( Bus Enable )  데이터교환을 할 수있는 데이터버스 / 주소버스 를 활성화 한다.  RWB ( Read Writh )  BE핀이 활성화가 된상태에서  메모리의 데이터를 읽어오고,  메모리에 저장하는 역활 또는 외부장치와의 통신 | |  | | 클럭 (오실레이터)  1초에 CPU의 명령사이클 실행 횟수를 정한다.  1헤르츠( Hz ) =1초에 1번씩 실행  10000MHz => 1초에 100만번 실행 | |  | | VCC 전원  GND 접지  OutPut  Hz에 해당하는 횟수만큼 Low 과 High  신호를 보낸다 | |  | | 음극 에서 전자가 발생하고 - => +  양극 에서 전류가 발생한다. + => - | |  | | ROM ( Read Only Memory )  전원이 꺼져도 데이터가 유지되는대  읽기전용 이므로 한번 데이터 입력을 해놓으면  수정이 불가하다.  데이터 덮어쓰는 방식으로 데이터수정이 가능한 롬  EROM( CY62256 X8비트 )  ,EEROM( AT28256 X8비트) | |  | | ROM 핀 데이터시트  WEB ( Write Enable)  데이터를 쓰거나 변경하는 것을 허용 한다  OEB( Output Enable )  데이터 출력을 활성화.  CEB( Chip Enable )  Rom의 활성화.  Cput의 15번핀과 연결하면 클럭에 의해 명령사이클이 돌때 프로그램의 주소벡터를 참조하여 실행한다.  A14  롬은15핀 CPU는16핀 인대  보편적으로 롬의 어드레스 최상위핀은 주소벡터이다 Hight면 Cput가 실행하는 프로그램의 실행위치 주소 이다. Low일 경우 Ram을 사용 한다. | |  | | RAM ( Random Access Memory ) | |  | | Ram 핀 데이터시트  OEB  Cpu의 요청이 있을떄 데이터를 가져올라면  활성화가 되어 있어야겠다  WEB  Cpu가 Ram에 데이터를 저장하려할때  활성화가 되어 있어야겠다  I/O  데이터를 주고 받고 하는 핀  OEB WEB의 활성화에 따라  Cpu의 RWB 를 Ram의 WEB와 버스 연결  Cpu의 최상위 어드레스핀이 0이면 Ram을 사용한다  1이라면 Rom을 사용한다 | |  | | - Binary Code -  (OpCode)[ operation code / Operand ]  실행 코드 / 피연산자  - 어셈블리어 -  니모닉 / 피연산자 | |  | | 주소지정 기호 $ 의 피연산자는 2바이트를 차지한다  직접대입 하는 상수 기호 # 의 피연산자는 1바이트를 차지한다 | |  | | 니모닉+주소지정 모드(Addressing mode) = OpCode  니모닉은 주소지정 모드에 따라  같은 니모닉 이라도 다르게 OpCode가 결정된다 | |  | | - A 모드( A레지스터 ) -  ALU의 계산 결과 값이 들어가는 레지스터  1 바이트  실행사이클 2 | |  | |  | |  | |  | |  | | - 명령어싸이클 - | |  | | - asl의 실행 사이클 - | |  | | - Cpu 파이프라인 -  명령어수준 병렬성( Instruction Level Parallelism )  하나의 명령어안에 있는 명령어를 병렬로 실행 한다.  각 단계는 병렬로 수행되고, 한 번에 여러 명령어를 처리할 수 있게 한다 | |  | | - 6502 파이프라인 -  내부연산( ALU ,Controll Unit )  Cpu레지스터에 Load되어있는 명령어를 연산한다  외부연산( Ram )  Ram에서 명령어를 Cpu내부의 레지스터에 Load한다  Cpu사이클이 1번 실행 될때 내부연산 과 외부연산 이 한번씩 실행 | |  | | - A모드( A레지스터 ) 명령어 실행사이클 -  1 사이클  Fetch[ 0A ] Decode[ ] Execute[ ] WriteBack[ ]  2 사이클  Fetch[ 4A ] Decode[ 0A ] Execute[ ] WriteBack[ ]  3 사이클  Fetch[ 4A ] Decode[ ] Execute[ 0A ] WriteBack[ ]  4 사이클  Fetch[ 6A ] Decode[ 4A ] Execute[ ] WriteBack[ ]  5 사이클  Fetch[ 6A ] Decode[] Execute[ 4A ] WriteBack[ ]  6 사이클  Fetch[ 2A ] Decode[ 6A ] Execute[ ] WriteBack[ ]  7 사이클  Fetch[ 2A ] Decode[ ] Execute[ 6A ] WriteBack[ ]  .  ( 1.Fetch 2.Decode 3.Execute 4.WriteBack )  첫 명령어 실행까지 3사이클이 걸리지만  그이후로 실행 사이클은 2회가 된다  니모닉먼저 읽고 피연산자 순서로  실행사이클은 돌게 된다  하지만 A모드의 경우 피연산자가 없기때문에  니모닉 다음으로 Fetch한 데이터는 피연산자가 아니기 때문에 Fetch값을 버리게 된다 그다음 실행사이클에 다음 명령어를 읽어 오게 된다.  주소버스[ 명령어주소 니모닉+피연산자 ]  PC++[ 다음실행할 명령어의 주소 ]  증가하면 다음 명령어 실행  증가하지 않으면 대기하게되고 1사이클이 허비된다  디코딩( Decode )  어셈블리어니모닉을 바이너리코드로 변환하는 과정  인코딩 ( Incode )  바이너리코드를 니모닉으로 사람이 알수있는  형태로 변환 하는 과정  offset  데이터나 메모리주소를 말한다  opcode  니모닉 을 말한다  Pc++( Program Counter )  다음 명령어 주소를 말한다 | |  | | - # 즉시모드 ( Immediate Mode ) -  니모닉 #상수  데이터 핀이 8개 있으닌깐  데이터 레지스터는 8비트 가 된다  니모닉 1바이트 #상수 1바이트  총 2바이트  즉시모드 명령어 사이클  8000번 주소의 16진수 A9는 니모닉 lda의 16진수  8001번 주소의 0A는 피연산자 데이터값  1 사이클  Fetch[ A9 ] Decode[ ] Execute[ ] WriteBack[ ]  2 사이클  Fetch[ 0A ] Decode[ A9 ] Execute[ ] WriteBack[ ]  3 사이클  Fetch[ 69 ] Decode[ ] Execute[ A9 0A ] WriteBack[ ]  4 사이클  Fetch[ 06 ] Decode[ 69 ] Execute[ ] WriteBack[ ]  5사이클  Fetch[ E9 ] Decode[ ] Execute[ 69 06 ] WriteBack[ ]  .  .  .  1.Fetch 하고  2.Decode에서 니모닉이 즉시모드인걸 알게되고  3.Execute할때 Fetch에있는 피연산자를 같이 실행 | |  | | - $( a ) 절대주소 모드 -  6502 Cpu의 주소핀은 16개 이다  그렇기 떄문에 주소길이는 16비트가 된다  그렇다면 32비트 Cpu의 주소핀은 32개 이기때문에  한번에 처리할 수 있는 명령어 길이는 32비트 이다  만약 8비트 주소핀에서 16비트의 주소를 읽어야  한다면 2번에 나눠서 읽어온다  Store뜻 -> 값을 저장한다  니모닉 + 주소값  ( 1바이트 ) + ( 2바이트 ) 총 3바이트  1 사이클  Fetch[ 6D ] Decode[ ] Execute[ ] WriteBack[ ]  2 사이클  Fetch[ 0A ] Decode[ 6D ] Execute[ ] WriteBack[ ]  3 사이클  Fetch[ CB ] Decode[ 0A ] Execute[ 6D ] WriteBack[ ]  6D가 디코딩 되면 피연산자가 2바이트 라는걸  알고 Fetch에 우선 낮은바이트가 피연산자로  오고 다음사이클에 높은바이트가 들어온다  여기서 피연산자를 읽어오는대 2사이클이 걸린다  명령어를 실행하는대는 3사이클이 걸린다  4 사이클  Data는 주소버스( CB0A 의 값 )  Fetch[ Data ] Decode[ ] Execute[ ] WriteBack[ ]  Data를 가져오는대 까지는 4 사이클이 걸린다  5 사이클  Fetch[ 8D ] Decode[ ] Execute[ Data ] WriteBack[ ]  6 사이클  Fetch[ 00 ] Decode[ 8D ] Execute[ ] WriteBack[ ]  첫 명령어 실행까지 5사이클이 걸리지만  그이후로 실행 사이클은 4회가 된다 | |  | | - zp( zero page ) 주소지정 모드 -  0x 는 뒤에 16진수라는 접두사 이다  1 바이트의 주소 범위는 0x00 ~ 0xFF( 2의8승 256 )  이범위를 페이지 라고 한다.  사실 메모리상 표시는 0x0000 ~ 0x00FF 인대  6502의 주소 범위는 주소핀이 16개 닌깐  2의16승 65536이고 16진수로 FFFF 이다  0x00 ~ 0xFFFF  65536( 6502 주소범위 ) / 256( 페이지 ) =  256페이지라고 할 수 있다  이걸 Page0 Page1 Page2 Page3 Page4 Page5 . .  이런식으로 부른다  첫번째 페이지를 제로페이지 라고 한다  니모닉 + 1 바이트( 0x0000 ~ 0x00FF )  1 사이클  Fetch[ 65 ] Decode[ ] Execute[ ] WriteBack[ ]  2 사이클  Fetch[ 0A ] Decode[ 65] Execute[ ] WriteBack[ ]  3 사이클  Fetch[ Data ] Decode[ ] Execute[ ] WriteBack[ ]  4 사이클  Fetch[ 85 ] Decode[ ] Execute[ ] WriteBack[ ]  5 사이클  Fetch[ 00 ] Decode[ ] Execute[ 85 ] WriteBack[ ]  첫 명령어 실행까지 4사이클이 걸리지만  그이후로 실행 사이클은 3회가 된다  주소비트 길이는 6502의  주소핀16개닌깐 16비트 인대  피연산자의 주소가 1바이트일경우  하이( High )바이트는 00이 된다 | |  | | - 메모리 페이지 레이아웃 -  Page 0  제로페이지 주소지정 모드 에서 사용한다  Page 1  프로세서 스택메모리로 사용한다  각 페이지는 0x0000 ~0x00FF 인대  스택메모리로 사용되는 페이지는  하이바이트를 1로 강제 하기때문에  0x0100 ~ 0x00FF 이다 | |  | | - 연산자 -  니모닉 명령어를 함수라고 생각하면 된다  a절대주소 모드  받아올림 초기화  시작을 0 으로 한다는 말 | |  | | zp주소지정 모드 의 경우 | |  | | - 데이터 저장 -  모든 데이터에는 메모리 주소가 있다    데이터읽기( load )  특정 주소로 가서 데이터를 읽어옴  테이터쓰기 ( store )  특정 주소로 가서 데이터를 덮어씀  Cpu의 주소핀에서 주소버스를 타고  Ram의 주소핀에 접근한다  이때 접근한 주소에 값을 쓸껀지 읽어올 껀지  결정은 데이터버스의 신호에따라 결정 된다. | |  | | - 스택 -  스택으로 사용되는 메모리 범위  상위비트의 첫번째 비트를 1로 고정해서 사용  되는것이 일반적이다  이것은 스택영역의 방식이 상단에서 아래로 증가  하는 방식이기 때문이다  0x0100 ~ 0x01FF | |  | | - 상수 -  상수 표기법  16 진수 $  2 진수 %  ASCII ‘  주소를 표기할땐 헥스 를 사용하는게 좋다  상수에 별칭을 달아 사용할 수 있다  이미 메모리에 접근한다는 가정을 하고 있기때문에  상수를저장할 주소의 이름을 지정 할 수 있다 | |  | | - 데이터 블록 -  .BYTE 는 1바이트  .WORD 는 2바이트 | |  | | | |  | | - 라벨( label ) -  바이트 주소를 가르키는 별칭  고수준언어 goto의 실체  어셈블러에 의해 바이너리 변환시  실제 주소로 변환 된다  .ORG( origin ) 리셋벡터 설정  여기서 nums의 리셋백터는 바로아래  BYTE의 $00 이다  hello의 리셋백터는 $04 | |  | | - 6502의 레지스터 -  메인 레지스터 A  ALU의 입출력으로 사용되는 레지스터 이다  - 색인 레지스터 X Y -  어떤 배열의 주소를 X라 할떄 X+1 X+2 X-1  이런식으로 배열의 인덱스 번쨰줏소에 접근 가능  (비교값) cpx cpy  Compare 같은지 비교하는 비교 니모닉  8비트 데이터 임시저장  tax  a의 값을 x에 저장한다  txa  x의 값을 a에 저장한다  - 스택 데이터에 접근 -  txs  x의 값을 스택에 저장  tsx  스택의 값을 x에 저장  색인레지스터 S  스택 0x0100 ~ 0x01FF 의 주소를 가르는 포인터  스택의 시작 주소는 0x01FF 이고  Push / Pop( Pull )  Push할떄마다 주소번지가 줄어들고  Pop할때마다 주소가 늘어난다 | |  | | Pha  Push해라 a레지스터에 있는 값을  Pla  Pull  해라 a레지스터에 있는 값을  jsr  함수안에서 서브루틴 호출시 다시 돌아와야  할 함수의 주소 스택에 저장한다  rts  스택에 저장된 서브루틴 함수의 주소로 이동한다  서브루틴  프로그램 안에서 호출되어 실행되는 함수 또는  코드 블럭 이다  - 프로세서 상태 레지스터 P -  PC ( 프로그램 카운터 )  현재 Fetch한 명령어의 주소를 저장한다 | |  | | - i 묵시적 주소지정 모드 -  피연산자를 적지않았어도 내부적으로 이미 알고있는  니모닉  nop ( EA )  아무런 연산도 하지않는다  프로그램 실행에 딜레이가 필요할때 사용 한다  - 딜레이를 넣고싶을때 -  또는 메모리공간을 nop로 차지(예약)시켜놓고  기능을 하나하나 구현하는 하기도 한다. | |  | | - 프로그램 종료 -  6502에서 사용하는 일반적인 메인뤂프형태 는  임베디드 시스템의 기본상태의 무한루프 이고  전원을 공급하지 않을때 까지 무한루프를 실행한다.  예를 들어 냉장고의 전원을 켰을때 계속해서  실행되고 있는것  한번실행후 더이상 실행하고 싶지않으면  본인 레이블로 점프하여 루프돌게 하면된다  이것을 6502에서의 프로그램 종료를 뜻한다. | |  | | - 정리 - | |  | | - 산술 연산 - | |  | | - adc ( Add With Carry ) -  두 8비트의 정수의 덧셈  A레지스터[ ]안의 데이터 + 피연산자( 메모리[ ] )  A = A+M  문제 1 받아올림에 의해서 바이트자릿수를  넘길경우 A레지스터에는 어떤값이 저장되는가  주어진 비트로 표현 가능한 범위를 넘어가는 경우  정수 오버플로우 라고 한다  이것은 최솟값의 범위를 넘었을경우 높은  자릿수로 넘어가는 것도 오버플로우 라고 한다  정수오버플로우가 될경우  최댓값은 최솟값으로 넘어가고  최솟값은 최댓값으로 넘어간다  멀티바이트 덧셈  하위바이트에서 최상위비트에 받아올림이 생겼을  경우 그 받아올림수를  상태레지스터의 C( Carry) 비트에 저장 하고  상위바이트에 Carray비트를 같이 더한다.  Carry받아올림 비트를 사용하는 니모닉  adc 명령어는 상태레지스터의 NVZC  에 모두 영향을 줄 수 있다.  두 수의 덧셈중 음수의 경우  부호비트에 의해 C플러그가 항상 1이되기때문에  C플래그 하나만 보고는 받아올림에 의해  C플래그가 1이된것인지 확신할 수 없다 | |  | | - 2배 정밀도 덧셈 -  두 수를 더하기전에 플래그C비트를 지워줘야  원하는 연산 결과를 얻을수 있다  clc를 사용하면 플래그C비트가 0으로 초기화 된다 | |  | | - clc -  C플래그를 0으로 설정  두 수의 덧셈중 음수의 경우  부호비트에 의해 C플러그가 항상 1이되기때문에  C플래그 하나만 보고는 받아올림에 의해  C플래그가 1이된것인지 확신할 수 없다  때문에 V플래그가 확실한 오버플로우 플래그이다 | |  | | UnSigned  부호비트가 없으며 모든 비트가 숫자를 표현하는대  사용된다 따라서 8Bit일경우 0부터255까지 표현가능  Signed  부호비트를 사용하며  1 인경우 -0 또는 음수  0 인 경우 +0 또는 양수 이다  8Bit일 경우 -128에서 127까지 표현가능  UnSigned정수는 Carray플래그로 오버플로우 확인  Signed정수는 V플래그로 오버플로우 확인 | |  | | - 오버플로우 플래그 V -  7비트인 Signed정수의 오버플로우 일 경우  V플래그는 1 이 된다  Signed타입에 오버플로우가 발생하면  V플래그가 발생하고  UnSigned타입에 오버플로우가 발생하면  C플래그가 발생한다  1000 과 1000의 연산을 하면  받아올림이 100이다 이때  1 은 캐리아웃 그옆의 0 은 캐리인  이라 할때 캐리아웃과 캐리인이 같지 않으면  V플래그가 1 이되는 오버프로우 상태고  캐리아웃이 1이기때문에 C플래그도 1이된다  간단히말하면 이렇게도 말할 수 있다  두 수의 부호가 같을때 오버플로가 발생한다  Signed는 부호가 같은 두수의 연산중  부호비트( 7bit ) 가 서로 같은대 연산을 한다음  부호비트가 바뀌었다면 V플래그가 켜진다.  Signed  부호비트가 동일한 두수가 연산후 7Bit자리에서  받아올림에 의해 부호비트가 변경되면 오버플로  가 발생하여 V플래그에 저장된다  UnSigned  비트자리수를 넘어서는 받아올림이 생긴경우  C플래그에 저장된다  이것은 두 음수 의 연산의 경우 무조건 받아올림이  생기기때문에 오버플로우를 판단하는 지표라  할 수 없다 | |  | | - Z( Zero ) 플래그 -  8비트가 모두 0 이면 Z플래그가 켜진다  조건문 사용시 ~가 0 일대 명령어를 실행해라  라는 비교를 할때 사용된다  니모닉 실행했는대 a레지스터에 결과값이 0  이라면 켜진다  - N( Negative ) 플래그 -  7Bit가 모두 1일때 N플래그가 활성화 된다. | |  | | - sbc -  캐리비트를 이용한 뺄셈  - sec -  뺄셈을 하기전에 항상 호출 해줘야 한다.  C플래그를 1로 Set 해주는 니모닉 이다.  C플래그가 1이면 받아내림이 없었다는 거고 양수 라는 뜻이다 C플래그가 0이면 받아내림이 있었고 결과값이 음수라는 뜻이다  sbc 의 양수 계산  sbc니모닉은 하드웨어적으로 두단계로 실행이 된다.  1) 1의보수  2) sec으로 C플래그가 1이된걸 더하여  2의 보수가 된다 | |  | | sbc 의 음수 계산 | |  | | - 16 비트 양수 뺄셈 - | |  | | - 16 비트 음수 뺄셈 - | |  | | - 증감 연산 -  ++i; ,--i; 와 같은 니모닉  X,Y레지스터는 색인레지스터이기 떄문에  +1씩 증가시킬 일이 많기때문에 하드웨어 제작자가 니모닉을 따로 만들어 준것이다  inc ( Increment 증가 )  원하는 레지스터의 값을 1 증가시킨다  INC $00FF 메모리 주소 $00FF에 저장된 값을 1씩 증가시킨다.  inx  x레지스터의 값을 1 증가시킨다  iny  y레지스터의 값을 1 증가시킨다 | |  | | - 색인 주소지정 모드 -  ??, x ??,y 콤마하고 색인주소지정 모드가 나오면  피연산자에 콤마뒤에 나오는 색인주소값을 더한다 | |  | | 색인주소 지정모드 와 절대 주소지정 모드의  사이클은 똑같다 | | \ | | - a,x / a,y 주소지정 모드 -  a,x / a,y 주소지정모드는 기본적으로 4사이클  이지만 페이지 크로싱이 있다면 5사이클이 된다 | |  | | - 페이지 크로싱 ( Page Crossing ) -  베이스주소 nums가 0xC0FC일때  X가 값이 2 일때  페이지를 넘어가지않아 페이지 크로싱이 없기  때문에 4사이클 그대로 이다  x가 6일때  페이지가 넘어가게 되어 cpu내부적으로  다음페이지에 있는 주소를 가져와야하는  행동을 해야하기 때문에 사이클수가 1 증가한다  이것을 페이지 크로싱 이라한다  주소연산 결과가 다음페이지를 가리키는 현상 | | 페이지가 바뀌지 않는 경우 | | | | 페이지가 바뀜 | | | | zp,x 주소지정 모드 | | | |  | | - zp,x 래핑 ( Wrapping ) -  래핑의 뜻은 반복 / 루프 를 뜻한다 | |  | | - 증감 연산의 플래그 -  A레지스터를 증감할경우 어던 플래그가 켜질까  오버플로의 정의는  산술연산의 열과가 범위를 넘을떄 발생한다  그러나 증감연산은 산술연산이 아니기때문에  오버플로우가 없다.  증감연산은 N,Z플래그만 바꾸고  산술연산은 N,Z,V,C도 바꾼다 | |  | | - 비트 이동( Shift ) -  이동 ( Shift )  새로 들어오는 비트는 항상 0  밀려나간 비트는 C플래그로 들어간다  회전 ( Rotate )  새로 들어오는 비트는 C플래그의 값  왼쪽이나 오른쪽으로 전체비트를 한칸씩 이동 한다  asl  전체 비트를 왼쪽으로 민다  새로채워지는 비트는 0 이다 | |  | | | |  | | asl 이동시 밀려난 비트는 C플래그에 저장되지만  어떤용도로 사용되지는 않는다  - lsr -  전체 비트를 오른쪽으로 한칸씩 민다  새로채워지는 비트는 0 이다  N플래그는 항상 0 이된다  최상위 비트가 뭐였던간에 0이들어가닌깐  언제나 양수 이다  - rol -  비트회전  전체 비트를 왼쪽으로 한칸씩 민다  밀려나간 비트는 C플래그에 저장된다  새로채워지는 비트는 C플래그에 들어있는 값이다 | |  | | - 논리연산 -  a레지스터에 있는 값과 연산하고 a레지스터에 결과값을 저장한다  곱셈 / 나눗셈 은 따로없기때문에  더하기 니모닉을 이용해서 직접 구현해야 한다 | |  | | - 6502 데이터 타입 -  cpu에서 보내는 바이너리코드 신호를  윈도우 드라이버가 중간에 필요한 신호로 변환  시켜서 원하는 결과를 실행시키게끔 해준다  6502의 데이터 너비는 8비트  메모리에 저장된 비트가 전부기 떄문에  엄밀히 데이터형식은 없다  byte  6502가 한번에 처리하는 명령어 크기  이기도 하고 레지스터의 크기도 8비트이기  때문에 바이트형은 있다고 볼 수도 있다  ( ALU의 Word( 2바이트 ) 크기 )  char  아스키 코드 이것을 아스키 코드를 해석하기  위해 출력장치에 어떤 매핑이 있어야하는대  그것을 운영체제드라이버가 해준다 | |  | | - 실수형 -  6502의 cpu alu의 기초는 정수형 처리이며  요즘의cpu는 실수형을 지원(별도의 회로가탑제)한다  6502에서 실수형은 직접 만들어서 써야한다 | |  | | - 형변환 -  고수준 언어의 형변환시 묵시적 으로  가능한이유는  그냥 대입해도 정보를 잃지않을때 허용된다  비트크기가 작은곳에서 큰곳으로 변환시  묵시적 으로 가능  바이트가 큰걸로 옮겨졋을때 로우바이트는 그대로  하이바이트는 0으로 기본셋팅 된다  명시적 으로 형변환을 해야할때  넓은공간에서 좁은공간으로 변환할때  명시적으로 한다  저장공간이 좁아지기 때문에 정보손실의 위험이  있을 수 있다 그렇기때문에 너정말 변환 할꺼야  라고 컴파일러가 묻는것이다  하이바이트는 날아가고 로우바이트만 갖게된다  형변환 이란  비트패턴은 변하지않는다  그 비트패턴을 다르게 해석할 뿐이다 | |  | | - < , > 단항 바이트 연산자 -  <  하위 바이트 반환  1234 = 34  >  상위 바이트 반환  1234 = 12 | |  | | - 조건문 -  cmp ( Compare )  A - M 수행  N, Z ,C 플래그만 변경한다  결과값을 저장하지 않는다  주어진 비교값이 a레지스터와 같으면  Z플래그가 켜진다  주어진 값보다 레지스터값이 더 크면  C플래그가 켜진다  일반적으로 cmp는 UnSigned비트 비교만 가능한대  각 플래그에따라 분기문을 실행 시킨다  - 분기 니모닉 -  beq  같으면 점프해라  bne  같지않을때 점프해라  bcc  캐리비트가 클리어일때 C가 0 일때 점프해라  bcs  캐리비트가 셋 됬을때 1 일때 점프해라  산술연산 결과값을 가지고 분기문을 사용할때  사용하는 니모닉  bmi  음수 플래그가 1이면 분기  bpl  bvc  bvs | |  | | cmp의 Signed비트 비교방법  sbc를 사용해 V플래그를 이용한다  a-m을해서  134.  그러나 6502cpu 이후 Signed비트와 UnSigned비트  의 비교하는 분기니모닉이 있다 | |  | | - r ( Relative ) 상대 주소지정 모드 -  오프셋  현재 위치를 기준으로 타겟으로하는  메모리위치에 접근할 수 있는 거리의 범위  이를 통해 현재위치에서 타겟위치까지  상대적으로 얼마나 떨어져있는지 알 수 있다  16비트의 주소버스를 가지고있지만  명령어의 주소나 레지스터의 오프셋은 8비트로  제한 되어 있다  r 의 오프셋은 8비트 라서 앞뒤로 127 바이트  점프가 가능하다  라벨을 사용하면 자동으로 상대주소를 계산해준다 | |  | | - jmp -  16 비트주소를 갖는다  PC레지스터에 | |  | | | |  | | 반복문의 색인 비교에 적합 cmp와 거의 같음  cpx  cpy | |  | | - bit -  a레지스터의 값을 바꾸지않고  and 비교를 한다 모든비트가 0이면  Z플래그가 1 로 켜진다  메모리의 bit7은 N플래그에 들어가고  메모리의 bit6은 V플래그에 들어간다  이후 다양한 응용을 통해 분기를 한다 | |  | | | | - 상태 플래그표 - | | | |  | | - 반복문 -  반복문 이란 고수준에 있는 개념이다  기계어에서는 반복문의 니모닉은 없다  while문을 어셈블리어로  1부터 20까지 합하기  inx를 이용해  x 색인을 증가 시킴  더 빠른 1부터 20까지 합하기  dex를 이용해  20부터 시작해 감소하면서 더하기 | |  | | - 나눗셈 - | |  | | - 배열 -  요소들이 차례대로(순서대로) 저장 되어있다  n번째요소 주소 =시작주소 + 자료형의크기 x 색인  위의 배열의 색인주소를 찾는 공식에 의해  첫번째 인덱스가 왜 0부터 시작하는지에 대해  알수가있다  int[ 3 ]  0x4000 + 4 x 3 => 4012번째 주소 | |  | | 배열의 합구하기 | |  | | 배열의 최대값 찾기 | |  | | 배열의 문자위치 찾기  더 빠른 배열의 문자위치 찾기  문자열의 마지막에 NULL문자로 끝을 표현  NULL은 바이트로 0 이다 | |  | | 배열의 문자열 길이 구하기 | |  | | 문자열 복사하기  dst=$80  페이지0에 복사하면 1 바이트 수 가 줄어든다  6502 CPU 의 실행할 코드들은 Rom에 저장  되어있다가 CPU가 실행할때 RAM으로 복사되어  CPU가 RAM 에 복사된 코드를 실행한다  요즘 CPU는 RAM에 바로 저장되어 있다 | |  | | 배열의 문자요소를 소문자로 바꾸기  아스키코드 를 이해하고 있는지에 대한 문제  이며 소문자로 바꾸기 위해선 롬에있는  문자열을 램에다 복사를 해야한다  str에 저장된 Hell World는 Rom에 저장되어  있기때문에 덮어쓸 수 가없다  Rom의 WEB ( Write Enable)을 키면 되겠지만  켜지않은 상태라 가정했을때  문자열은 const 읽기전용 Read Only이다  데이터섹션에 할당되어 사용된다  상황에따라 동적으로 힙섹션에 할당하기도 한다  그렇기때문에 문자열을 바꾸려하면 오류가 난다 | |  | | - 다차원 배열 -  x,y요소 주소 = 시작주소 + 1차원배열크기 \* y + x  결국 base주소 + offset으로 요소주소를 찾는대  고수준에서는 offset을 대신 계산 해주는것뿐이다  배열과 포인터는 같다 | |  | | 구구단156. | |  | | - 지시문 -  . ( Directive)  어셈블 중에 어셈블러가 뭘해야하는지 지시하는것 | |  | | - 함수 -  - 서브루틴 ( 프로시저 ,함수 ) -  프로시저  반환값이없는 함수를 프로시저 라고 한다  함수  반환값이 있는 함수를 함수라고 한다.  반환값이 있던 없던 상위개념으로 둘다 서브루틴  이라고 한다  회귀주소  리턴주소 서브루틴 실행후 다시 돌아와야 하는  주소를 서브루틴 실행하기 전에 넘겨줘야함 | |  | | zp에 회귀주소를 저장 | | jmp ( 절대 간접 주소지정 모드 ) | | | |  | | - 간접참조 ( Indirection ) -  - 역참조 ( Dereferencing ) -  간접참조 를 역참조 라고도 한다  피연산자에 있는 주소에 접근해서 값을 읽어오는 행위 | |  | | - 직접참조 -  어떤 변수에 접근해서 값을 읽어오는  행위 | |  | | 계산기 ( 룩업 테이블 버전 ) 162. | |  | | 계산기 ( 룩업 테이블 버전2 ) 163. | |  | | zp 회귀주소  함수체이닝, 재귀함수  함수에 데이터를 전달하는 대표적인 두가지 방법  zp 회귀주소의 문제점  함수체이닝을 할 수 없음  1) 함수 안에서 또다른 함수를 호출할 수 없음 | |  | | zp 회귀주소  zp 회귀주소의 문제점  2) 재귀함수가 동작하지 않음 | |  | | zp 회귀주소의 함수체이닝 해결방안 1.  각 함수마다 리턴주소를 따로둔다 | |  | | zp 회귀주소의 함수체이닝 해결방안 2.  스택을 사용하여 회귀주소를 저장 | |  | | zp 회귀주소의 재귀함수 해결방안  스택을 사용한 재귀함수 | |  | | - jsr ,rts -  호출과 회귀를 담당해주는 니모닉  jsr  함수를 호출 할때 사용  타겟함수로 점프  rts  함수에서 돌아올때 사용  스택사용 | |  | | zp 회귀주소의 함수체이닝 해결방안 2. 를  jsr ,rts를 이용하여 다시 해결하면 | |  | | jsr 의 함수호출 반환 동작표 | | jsr 상세동작표 | | | |  | | rts 의 함수호출 반환 동작표 | |  | | | |  | | 체이닝함수 일때 스택메모리의 상태  함수호출때 스택메모리에 푸쉬 되는건  리턴주소 인자값 반환값주소 그함수내에 사용하는 모든 지역변수  함수호출시 스택에 올라가고 그안에서  또다른 함수를 호출하면 스택메모리에 그위에 또  추가되서 올라가고 또올라가고 그런방식인대  여기서 호출이끝나면 스택에서 체이닝 역순으로  하나씩 pop된다 | |  | | - zp를 이용한 인자 전달 -  1) Page0을 중간 저장소로 사용  2) 스택을 중간 저장소로 사용  레지스터 수가 적은 6502는 Page0 방법을 쓴다  레지스터 대용으로 Page를 사용한것이다  ‘ . ’ 이붙은 라벨은  지역라벨 이라고한다  이런 지역을 정하는것이 .SUBROUTINE 이다  ‘ . ‘ 이 없는 라벨은 전역라벨 이라해서  어디서든 점프할 수 있다  .SUBROUTINE  라벨 범위 ( Scope ) 를 정하는지시문 | |  | | - 스택을 이용한 인자 전달 -  6502에서 A레지스터 에서만 스택에 값을  푸쉬 ,팝 할수있다 | |  | | pha  A레지스터의 값을 스택에 Push  스택포인트 1감소  pla  스택에서 A레지스터로 pop  스택포인트 1증가 | | pha 동작표 | | | | pla동작표 | | | |  | | 매개변수가 여러개일 경우  호출자 ( Caller )가 스택에 인자를 순서대로 넣어줌  호출규약en.wikipedia.org/wiki/X86\_calling\_conventions  컴파일러 ,언어 ,cpu에 의해 규칙을 정한다  함수 호출자와 함수 간의 규정을 해논것  함수가 인자를 찾을 수 있게 하기위해 만든것  인자는 어느 방향으로 스택에 넣을껀지 등 을 규정  일반적인 고수준 언어에서의 호출규약은  오른쪽에서 왼쪽으로 넣고 있다 | |  | | 스택 원상복구 방법  모든 인자를 스택으로 전달하면 좋은점  페이지0을 변수처럼 사용할 수 있다    현재함수 에서 사용중인 지역변수를 스택에 저장  +  함수호출 = 데이터 전달 + goto ( jsr ,rts )  +  회귀후 스택에 저장해논 지역변수를 다시 꺼내서  사용 | |  | | 함수의 반환값을 전달하는 방법  함수내부에서  rts하기전에 반환값을 전달해줘야하는대 그방법으론  1) zp를 사용  2) 스택을 사용  3) A ,X ,Y 레지스터 사용 | |  | | zp 를 사용한 반환값 전달 1.  덧셈 ,뺼셈 함수 | |  | | zp 를 사용한 반환값 전달 2.  덧셈 ,뺼셈 함수  함수 호출시에 반환값을 언제나 제로페이지00에  들어가게끔 규약을 잡으면 좋다 | |  | | 스택을 사용한 반환값 전달 2.  제일 범용적인 방법임 | |  | | A ,X ,Y 레지스터를 사용한 반환값 전달 | |  | | 반환값이 워드크기 이상일때 반환값 전달  스택 이나 페이지 를 사용한다 | |  | | 스택을 사용한 16 비트 반환값 전달  고수준 언어에서도 워드사이즈 보다  반환값이 커지는경우 이런 방식을 사용한다 | |  | | 고수준 언어에서는  스택을사용해 데이터를 주고 받는다  인자값 ,지역변수  ( 지역변수 최적화 플래그가 켜져있다면 레지스터에 저장 )  스택에 인자를 넣는것은 느리기때문에  x86 호출규약중 fastcall호출 규약은  인자가 8바이트까지는 레지스터를 통해 전달  인자가 많아지면 스택을 통해 전달하게 된다 | |  | | - 함수호출의 오버헤드 -  오버헤드  함수호출에 드는 비용  어떤 처리를 하기 위해 들어가는 간접적인 처리 시간 · 메모리 등을 말한다. 예를 들어 A라는 처리를 단순하게 실행한다면 10초 걸리는데, 안전성을 고려하고 부가적인 B라는 처리를 추가한 결과 처리시간이 15초 걸렸다면, 오버헤드는 5초가 된다. | |  | | 함수 미사용 했을때 비용( 오버헤드 )  3N | |  | | 스택 사용이 드는 비용 ( 오버헤드 ) | |  | | zp 사용시 드는 비용 ( 오버헤드 ) | |  | | 예외적으로 함수를 사용해야 하는 경우  함수로 만들려는 코드길이가 너무나 길어서  복붙할경우 코드양이 너무 방대해질 경우  즉 바이트수가 너무 많은 경우  3번이상 사용될 꺼같으면 함수를 쓰면  바이트수를 줄이 수 있다  그러나 3번이상 쓰인다고 모주건 함수로  만들기 보단 밸러스 에따라 판단을 해야한다 | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |
|  |