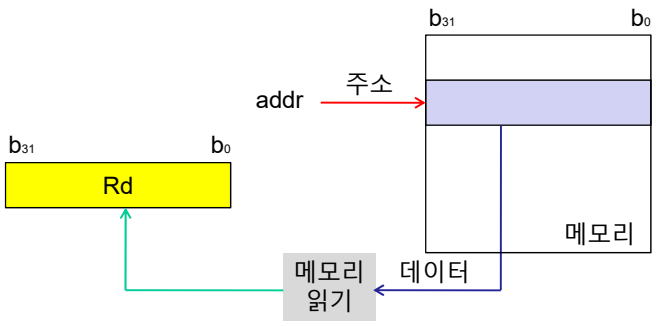
제4장
TOY 프로세서의 기계어와 실행

Youtube 주소
[4-1] http://youtu.be/YHYCumV-P0s
[4-2] http://youtu.be/RsSWcB8cnLE
[4-3] http://youtu.be/eOITR9Qu1Os

LOAD 명령어의 실행

- LOAD R1, value



직접 어드레싱의 기계어 표현

- 모든 명령어는 32비트로 표현

LOAD Rd, addr

OP	ALU	CC	DR	SR	V	SR2	VAL
001	000	111	Rd	111	1	PC 값과의 offset	

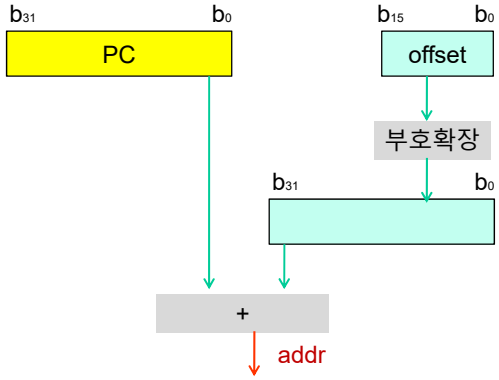
LOAD R1, value ; value의 주소는 현재에서 **+20**

OP	ALU	CC	DR	SR	V	SR2	VAL
001	000	111	001	111	1	0000 0000 0001 0011	

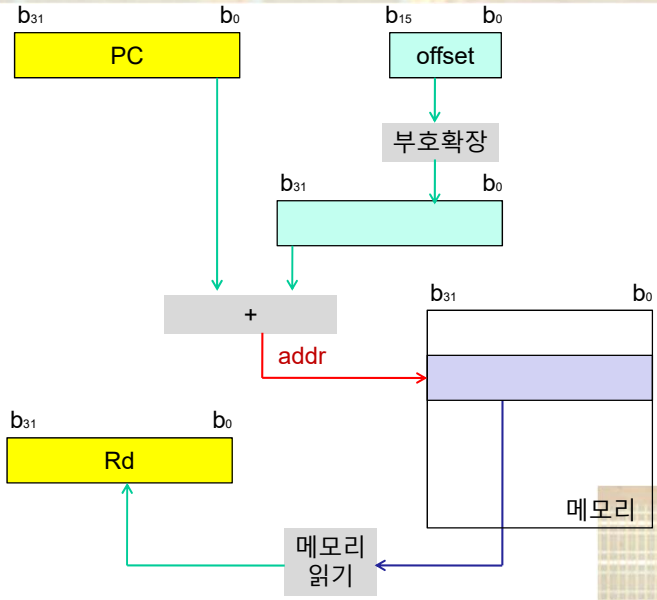
0x239F0013

LOAD / STORE / LEA / BR 에서 주소 결정

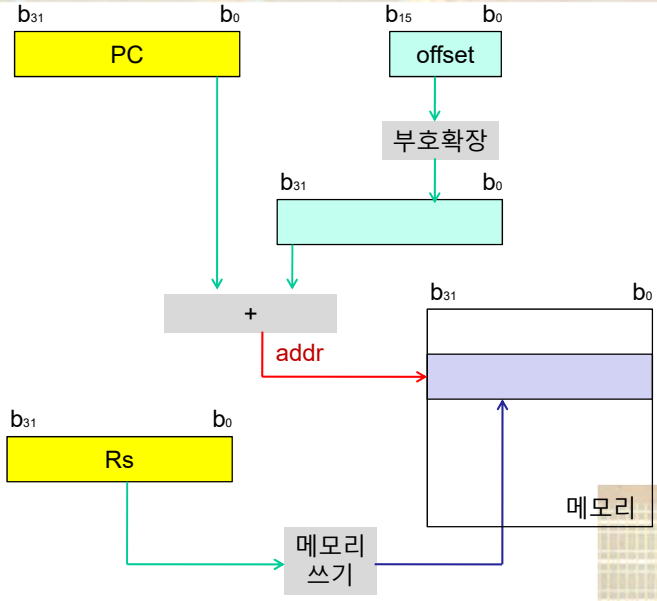
- 기계어 코드에서는 16비트 정수인 offset으로 표현
- 목적지 주소가 addr 이면, offset = addr – PC
- TOY 프로세서의 실행과정에서 주소 결정



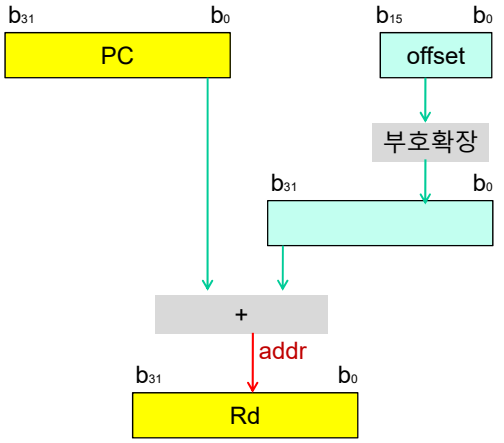
LOAD 명령어의 실행



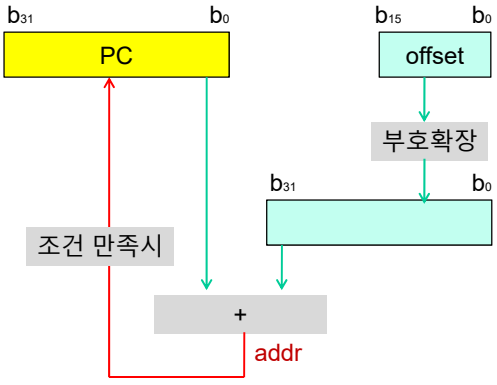
STORE 명령어의 실행



LEA 명령어의 실행



BR 명령어 실행



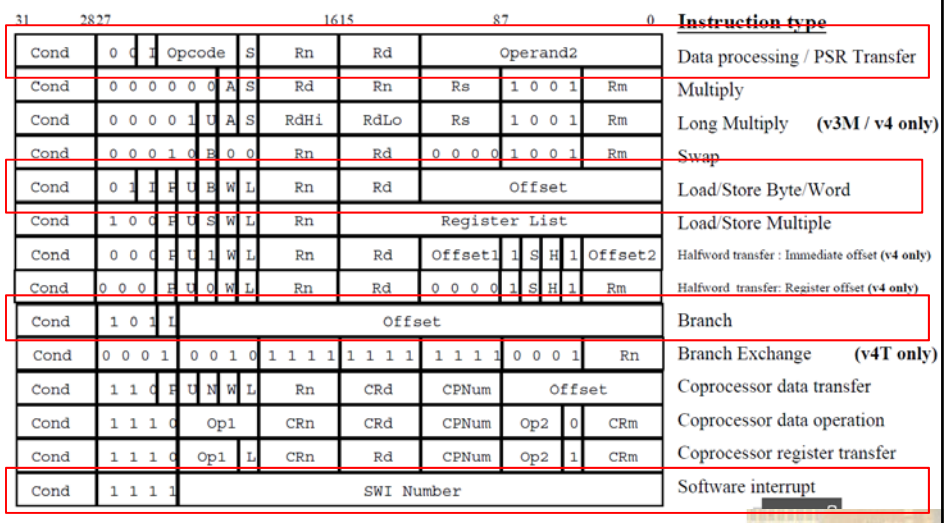
(참고) 레지스터 간접 어드레싱과 비교

- LOAD와 LDR의 비교
  - ✓LOAD: addr = PC + offset ← 현재 실행위치 기준
  - ✓LDR : addr = Ra + offset ← 지정된 레지스터 기준
  - → LOAD는 LDR에서 Ra 부분을 PC로 특정한 것
- STORE와 STR의 비교
  - ✓STORE는 STR에서 Ra 부분을 PC로 특정한 것
- BR과 BRR의 비교
  - ✓BR은 BRR에서 Ra 부분을 PC로 특정한 것

TOY 프로세서 기계어 코드

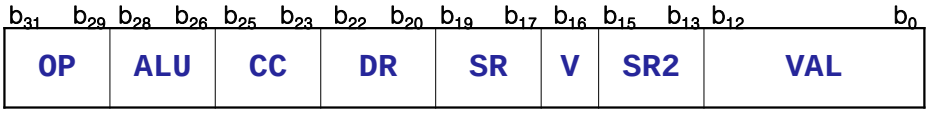
필드명	b_{31} b_{29} OP	b_{28} b_{26} ALU	b_{25} b_{23} CC	b_{22} b_{20} DR	b_{19} b_{17} SR	b_{16} V	b_{15} b_{13} SR2	b_{12} b_0 VAL
ADD	0 0 0	0 0 0	1 1 1	Rd	Rs	V		
SUB	0 0 0	0 0 1	1 1 1	Rd	Rs	V		
CMP	0 0 0	0 0 1	0 0 0	0 0 0	Rs	V		
AND	0 0 0	0 1 0	1 1 1	Rd	Rs	V		
OR	0 0 0	0 1 1	1 1 1	Rd	Rs	V		
XOR	0 0 0	1 0 0	1 1 1	Rd	Rs	V		
NOT	0 0 0	1 0 0	1 1 1	Rd	Rs	1	1 1 1	1 1 ..... 1 1
COPY	0 0 0	1 0 1	1 1 1	Rd	0 0 0	V		
LSL	0 0 0	1 1 0	1 1 1	Rd	Rs	1	0 0 0	0 n
LSR	0 0 0	1 1 0	1 1 1	Rd	Rs	1	0 0 0	1 n
ASL	0 0 0	1 1 1	1 1 1	Rd	Rs	1	0 0 0	0 n
ASR	0 0 0	1 1 1	1 1 1	Rd	Rs	1	0 0 0	1 n
LOAD	0 0 1	0 0 0	1 1 1	Rd	1 1 1	1		
LDR	0 0 1	0 0 0	1 1 1	Rd	Ra	1		
STORE	0 1 0	0 0 0	0 0 0	Rs	1 1 1	1		
STR	0 1 0	0 0 0	0 0 0	Rs	Ra	1		
BR	0 1 1	0 0 0	N Z P	1 1 1	1 1 1	1		
BRR	0 1 1	0 0 0	N Z P	1 1 1	Ra	1		
LEA	0 1 1	0 0 0	1 1 1	Rd	1 1 1	1		
LINK	0 1 1	0 0 0	1 1 1	1 1 0	1 1 1	1	0 0 0	0 0 ..... 0 1
RET	0 1 1	0 0 0	1 1 1	1 1 1	1 1 0	1	0 0 0	0 0 ..... 0 0
SWI	1 0 0	1 0 1	1 1 1	1 1 1	0 0 0	1	0 0 0	n
RTI	1 0 0	0 0 0	1 1 1	1 1 1	1 1 0	1	0 0 0	0 0 ..... 0 0
미정	1 0 1							

(참고) ARM 명령어 집합의 기계어 형식



Data Processing: I (Immediate), S (Set Condition)
Load/Store: L(Load), B(Byte)
Branch: L(Link)

TOY 기계어 32비트의 구성



- OP: 명령어 그룹 구분
  - ✓000: 산술논리연산, 001: 메모리 읽기, 010: 메모리쓰기, …
- ALU: 명령어 그룹 내에서 명령어 구분
- DR: 목적지 레지스터 지정 (STORE/SRR은 소스 레지스터)
- CC: 연산 결과를 DR에 기록하는 조건 (nzp 대응)
- SR: 연산의 소스 레지스터 또는 주소 레지스터
- V: SR2/VAL 부분의 구별 표시
  - ✓0: SR2 가 2번째 소스 레지스터 (VAL 부분은 무시)
  - ✓1: SR2/VAL을 합친 16비트가 정수로서 2번째 소스 오퍼랜드

ADD 명령어의 기계어 표현

ADD Rd, Rs, Rs2

OP	ALU	CC	DR	SR	V	SR2	VAL
000	000	111	Rd	Rs	V		

ADD R3, R1, R2

OP	ALU	CC	DR	SR	V	SR2	VAL
000	000	111	011	001	0	010	0 0000 0000 0000

0x03B24000

ADD R3, R1, 3

OP	ALU	CC	DR	SR	V	SR2	VAL
000	000	111	011	001	1	000	0 0000 0000 0011

0x03B30003

ADD/SUB/AND/OR/XOR 명령어의 기계어 표현

ADD/SUB/AND/OR/XOR Rd, Rs, Rs2

OP	ALU	CC	DR	SR	V	SR2	VAL
000	???	111	Rd	Rs	V		

SUB R3, R1, R2

OP	ALU	CC	DR	SR	V	SR2	VAL
000	001	111	011	001	0	010	0 0000 0000 0000

0x07B24000

AND R3, R1, 3

OP	ALU	CC	DR	SR	V	SR2	VAL
000	010	111	011	001	1	000	0 0000 0000 0011

0x0BB30003

ADD/SUB/AND/OR/XOR 명령어의 기계어 표현

ADD/SUB/AND/OR/XOR Rd, Rs, Rs2

OP	ALU	CC	DR	SR	V	SR2	VAL
000	???	111	Rd	Rs	V		

OR R4, R5, 0x1234

OP	ALU	CC	DR	SR	V	SR2	VAL
000	011	111	100	101	1	000	1 0010 0011 0100

0x0FCB1234

XOR R4, R5, R6

OP	ALU	CC	DR	SR	V	SR2	VAL
000	100	111	100	101	0	110	0 0000 0000 0000

0x13CAC000

CMP와 SUB의 기계어 비교

CMP R2, 9

OP	ALU	CC	DR	SR	V	SR2	VAL
000	001	000	000	010	1	000	0 0000 0000 1001

0x04050009

SUB R3, R2, 9

OP	ALU	CC	DR	SR	V	SR2	VAL
000	001	111	011	010	1	000	0 0000 0000 1001

0x07B50009

NOT 명령어의 기계어 표현

NOT Rd, Rs

OP	ALU	CC	DR	SR	V	SR2	VAL
000	100	111	Rd	Rs	1	111	1 1111 1111 1111

NOT R2, R0

OP	ALU	CC	DR	SR	V	SR2	VAL
000	100	111	010	000	1	111	1 1111 1111 1111

0x13CBFFFF

XOR R2, R0, -1

OP	ALU	CC	DR	SR	V	SR2	VAL
000	100	111	010	000	1	111	1 1111 1111 1111

0x13CBFFFF

COPY 명령어의 기계어 표현

COPY Rd, Rs2

OP	ALU	CC	DR	SR	V	SR2	VAL
000	101	111	Rd	000	V		

COPY R3, R2

OP	ALU	CC	DR	SR	V	SR2	VAL
000	101	111	011	000	0	010	0 0000 0000 0000

0x17B04000

COPY R3, 0x4321

OP	ALU	CC	DR	SR	V	SR2	VAL
000	101	111	011	000	1	010	0 0011 0010 0001

0x17B14321

LSL/LSR 명령어의 기계어 표현

LSL/LSR Rd, Rs, n

OP	ALU	CC	DR	SR	V	SR2	VAL	
000	110	111	Rd	Rs	1	000	R	n

LSL R3, R1, 2

OP	ALU	CC	DR	SR	V	SR2	VAL	
000	110	111	011	001	1	000	0	000 0000 0010

0x1BB30002

LSR R3, R1, 2

OP	ALU	CC	DR	SR	V	SR2	VAL	
000	110	111	011	001	1	000	1	000 0000 0010

0x1BB31002

ASL/ASR 명령어의 기계어 표현

ASL/ASR Rd, Rs, n

OP	ALU	CC	DR	SR	V	SR2	VAL	
000	111	111	Rd	Rs	1	000	R	n

ASL R3, R1, 3

OP	ALU	CC	DR	SR	V	SR2	VAL	
000	111	111	011	001	1	000	0	000 0000 0011

0x1FB30003

ASR R3, R1, 3

OP	ALU	CC	DR	SR	V	SR2	VAL	
000	111	111	011	001	1	000	1	000 0000 0011

0x1FB31003

LOAD 명령어의 기계어 표현

LOAD Rd, addr

OP	ALU	CC	DR	SR	V	SR2	VAL
001	000	111	Rd	111	1		

LOAD R1, value ; value의 주소는 현재에서 **+20**

OP	ALU	CC	DR	SR	V	SR2	VAL
001	000	111	001	111	1	000	0 0000 0001 0011

0x239F0013

LDR/LOAD 명령어의 기계어 표현

LDR R1, R6, 19

OP	ALU	CC	DR	SR	V	SR2	VAL
001	000	111	001	110	1	000	0 0000 0001 0011

0x239D0013

LOAD R1, value ; value의 주소는 현재에서 +20

OP	ALU	CC	DR	SR	V	SR2	VAL
001	000	111	001	111	1	000	0 0000 0001 0011

0x239F0013

STR/STORE 명령어의 기계어 표현

STR R1, R6, 19

OP	ALU	CC	DR	SR	V	SR2	VAL
010	000	000	001	110	1	000	0 0000 0001 0011

0x401D0013

STORE R1, value ; value의 주소는 현재에서 +20

OP	ALU	CC	DR	SR	V	SR2	VAL
010	000	000	001	111	1	000	0 0000 0001 0011

0x401F0013

분기 명령어 기계어 코드의 예

BR np, addr ; addr 은 다음 20번째 명령위치

OP	ALU	CC	DR	SR	V	SR2	VAL
011	000	101	111	111	1	000	0 0000 0001 0011

0x62FF0013

- PC + offset 의 값이 addr 의 위치가 되도록 offset 결정

BR 명령어의 기계어 표현

BR nzp, addr

OP	ALU	CC	DR	SR	V	SR2	VAL
011	000	NZP	111	111	1		

BR np, addr ; addr의 주소는 현재에서 +20

OP	ALU	CC	DR	SR	V	SR2	VAL
011	000	101	111	111	1	000	0 0000 0001 0011

0x62FF0013

BR p, loop ; loop의 주소는 현재에서 -5

OP	ALU	CC	DR	SR	V	SR2	VAL
011	000	001	111	111	1	111	1 1111 1111 1010

0x60FFFFFA

BRR/BR 명령어의 기계어 표현

BRR p, R6, -6

OP	ALU	CC	DR	SR	V	SR2	VAL
011	000	001	111	110	1	111	1 1111 1111 1010

0x60FDFFFA

BR p, loop ; loop의 주소는 현재에서 **-5**

OP	ALU	CC	DR	SR	V	SR2	VAL
011	000	001	111	111	1	111	1 1111 1111 1010

0x60FFFFFA

LEA 명령어의 기계어 표현

LEA Rd, addr

OP	ALU	CC	DR	SR	V	SR2	VAL
011	000	111	Rd	111	1		

LEA R1, value ; value의 주소는 현재에서 +20

OP	ALU	CC	DR	SR	V	SR2	VAL
011	000	111	001	111	1	000	0 0000 0001 0011

0x639F0013

SWI / RTI 명령어의 기계어 표현

SWI n

OP	ALU	CC	DR	SR	V	SR2	VAL
100	101	111	111	000	1	000	n

SWI 3

OP	ALU	CC	DR	SR	V	SR2	VAL
100	101	111	111	000	1	000	0 0000 0000 0011

0x97F10003

RTI

OP	ALU	CC	DR	SR	V	SR2	VAL
100	000	111	111	110	1	000	0 0000 0000 0000

0x83FD0000

두 수의 합 구하기 프로그램

```
  .ORIGIN 0x2000

  LOAD R1, x
  LOAD R2, y
  ADD R3, R1, R2
  STORE R3, z
loop: BR nzp, loop

x: .FILL 2
y: .FILL 3
z: .BLOCK 1
```


어셈블링 (Assembling)

- 메모리 주소별로 이진수 코드 결정
  - ✓0x2000 번지 부터 시작하여 기계어 코드들을 결정
- 레이블들은 실제 주소를 결정하여 처리
  - ✓loop: 0x2004
  - ✓x : 0x2005
  - ✓y : 0x2006
  - ✓z : 0x2007

```
  .ORIGIN 0x2000

  LOAD R1, x
  LOAD R2, y
  ADD R3, R1, R2
  STORE R3, z
loop: BR nzp, loop

x: .FILL 2
y: .FILL 3
z: .BLOCK 1
```

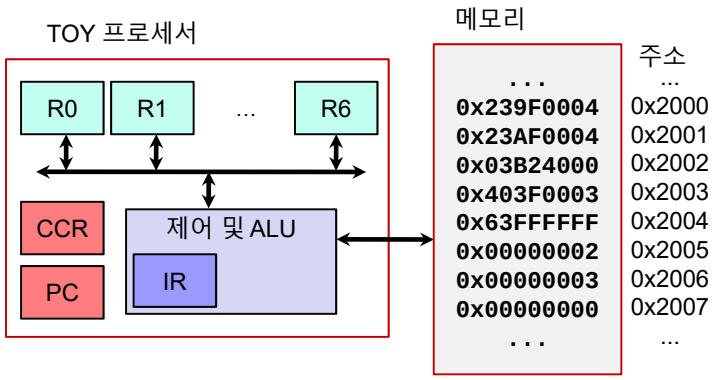
2수의 합 구하기 기계어 프로그램

주소	OP	ALU	CC	DR	SR	V	SR2	VAL	16진수	어셈블리언어
0x2000	001	000	111	001	111	1	000	0 0000 0000 0100	0x239F0004	LOAD R1, 0x2005
0x2001	001	000	111	010	111	1	000	0 0000 0000 0100	0x23AF0004	LOAD R2, 0x2006
0x2002	000	000	111	011	001	0	010	0 0000 0000 0000	0x03B24000	ADD R3, R1, R2
0x2003	010	000	000	011	111	1	000	0 0000 0000 0011	0x403F0003	STORE R3, 0x2007
0x2004	011	000	111	111	111	1	111	1 1111 1111 1111	0x63FFFFFF	BR nzp, 2004
0x2005	000	000	000	000	000	0	000	0 0000 0000 0010	0x00000002	.FILL 2
0x2006	000	000	000	000	000	0	000	0 0000 0000 0011	0x00000003	.FILL 3
0x2007	000	000	000	000	000	0	000	0 0000 0000 0000	0x00000000	.BLOCK 1

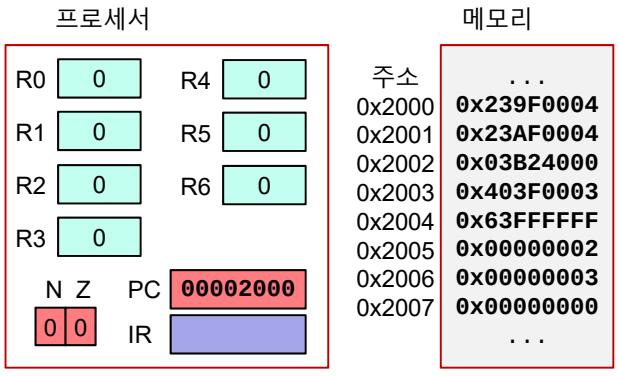
직접 기계어 코드로 작성하면…

```
.ORIGIN 0x2000
.FILL 0x239F0004
.FILL 0x23AF0004
.FILL 0x03B24000
.FILL 0x403F0003
.FILL 0x63FFFFFF
.FILL 0x00000002
.FILL 0x00000003
.FILL 0x00000000
```

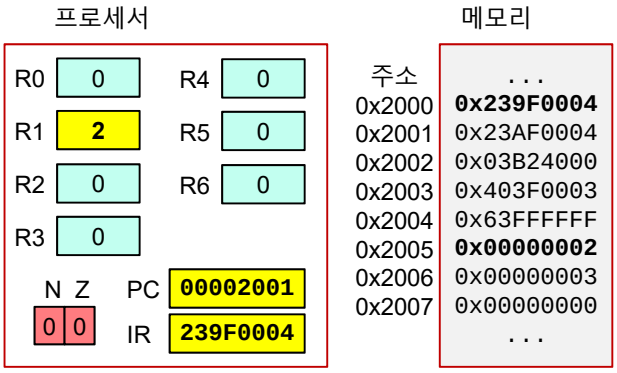
TOY 프로세서의 2수의 합 구하기 실행 과정



프로그램의 시작시점

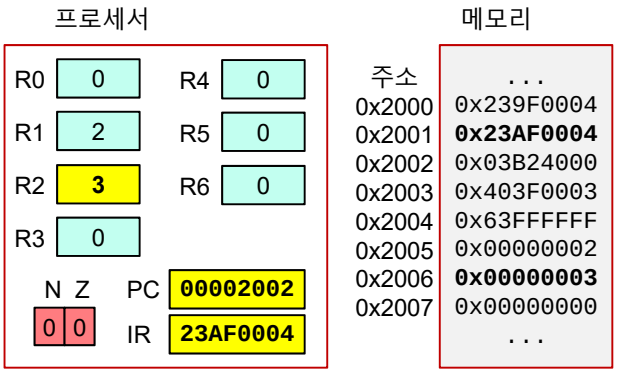


0x3000번지의 'LOAD R1, 0x2005' 실행 후



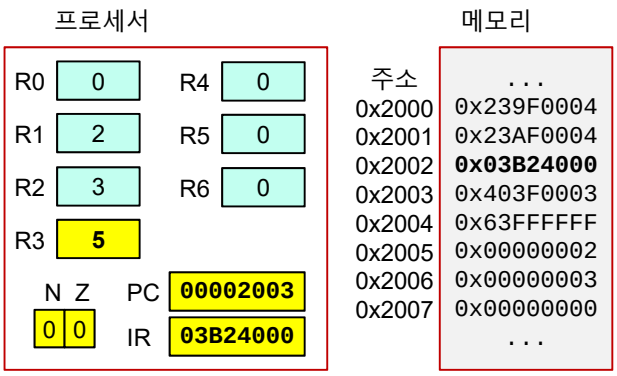
메모리 읽기/쓰기
(1) 0x2000 번지 읽기: **0x239F0004**
(2) 0x2005 번지 읽기: **0x00000002**

0x3001번지의 'LOAD R2, 0x2006' 실행 후



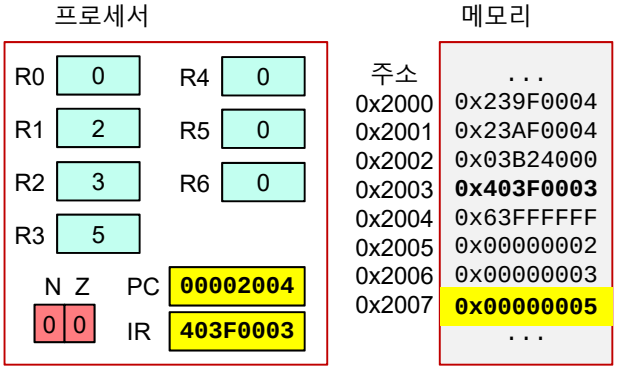
메모리 읽기/쓰기
(1) 0x2001 번지 읽기: **0x23AF0004**
(2) 0x2006 번지 읽기: **0x00000003**

0x3002번지의 'ADD R3, R1, R2' 실행 후



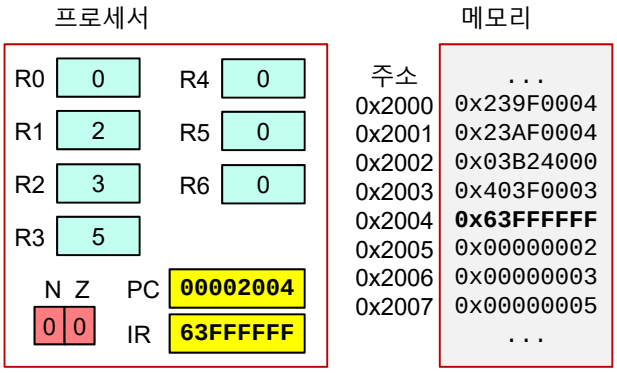
메모리 읽기/쓰기
(1) 0x2002 번지 읽기: **0x03B24000**

0x3003번지의 'STORE R3, 0x2007' 실행 후



메모리 읽기/쓰기
(1) 0x2003 번지 읽기: **0x403F0003**
(2) 0x2007 번지 쓰기: **0x00000005**

0x3004번지의 'BR nzp, 0x2004' 실행 후

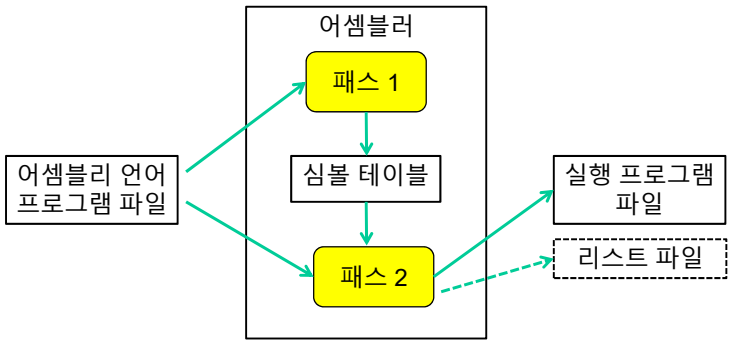


메모리 읽기/쓰기
(1) 0x2004 번지 읽기: **0x63FFFFFF**

어셈블러와 역어셈블러

- 어셈블러 (Assembler)
  - ✓어셈블리언어 명령어를 차례대로 대응되는 기계어 코드로 변환
  - ✓레이블들도 결정된 메모리 주소로 대체
  - ✓어셈블리 언어 프로그램 파일을 입력으로 받아서 기계어 코드로 된 실행 파일을 출력하는 프로그램
- 역 어셈블러 (Disassembler)
- 크로스 어셈블러 (Cross Assembler)

어셈블러의 실행과정 (2패스 어셈블러)



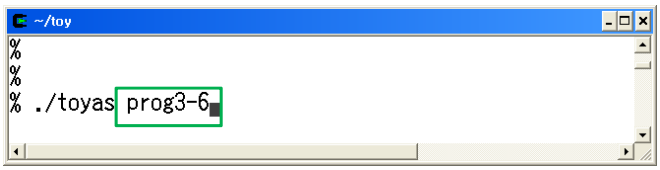
심볼 테이블

심볼	주소
loop	0x2004
x	0x2005
y	0x2006
z	0x2007

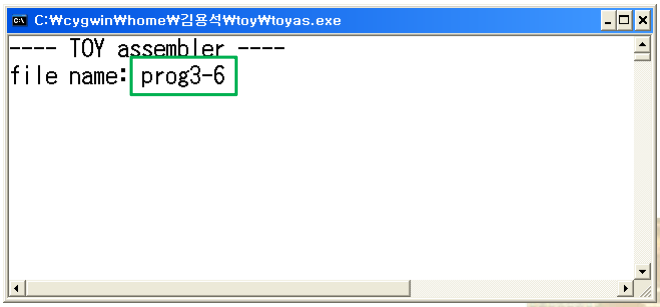
TOY 어셈블러 toyas의 실행

prog3-6.s 의 어셈블링

명령창에서 파일이름 입력



실행화면에서 파일이름 입력



Prog3-6.s

prog3-6 - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
        .ORIGIN 0x2000          ; program start address

        LOAD R1, value          ; load value into R1
        ASR R2, R1, 3           ; shift right 3 bits
        STORE R2, quot          ; store R2 on quot
        AND R2, R1, 0x07        ; mask the last 3 bits
        STORE R2, remain        ; store R2 on remain

halt:   BR nzp halt             ; branch here forever

value:  .FILL 0x12345           ; space for value with init. val.
quot:   .BLOCK 1                ; space quot without init. val.
remain: .BLOCK 1                ; space for remain without init. val.
```


Prog3-6.lst

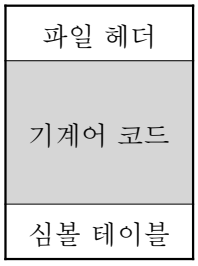
prog3-6 - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
Addr      Code          Line    Program
0x002000: 0x239f0005: (  3)     LOAD R1, value          ; load value into R1
0x002001: 0x1fa30003: (  4)     ASR R2, R1, 3           ; shift right 3 bits
0x002002: 0x402f0004: (  5)     STORE R2, quot          ; store R2 on quot
0x002003: 0x0ba30007: (  6)     AND R2, R1, 0x07        ; mask the last 3 bits
0x002004: 0x402f0003: (  7)     STORE R2, remain        ; store R2 on remain
0x002005: 0x63ffffff: (  9) halt:       BR nzp halt             ; branch here forever
0x002006: 0x00012345: ( 11) value:      .FILL 0x12345           ; space for value with init. val.
0x002007: 0x00000000: ( 12) quot:       .BLOCK 1                ; space quot without init. val.
0x002008: 0x00000000: ( 13) remain:     .BLOCK 1                ; space for remain without init. val

---- symbol table ----
    halt: 0x2005
   value: 0x2006
    quot: 0x2007
  remain: 0x2008
```

실행파일의 구조



예) TOY 파일포맷

```
0x544f5900
0x00002000
0x239f0005
0x1fa30003
0x402f0004
0x0ba30007
0x402f0003
0x63ffffff
0x00012345
0x00000000
0x00000000
```

TOY 파일포맷 매직번호

시작 주소

prog3-6.o의 내용

```
~/toy/example
% od -t x4 prog3-6.o
0000000 544f5900 00002000 239f0005 1fa31003
0000020 402f0004 0ba30007 402f0003 63ffffff
0000040 00012345 00000000 00000000
0000054
%
```

Cygwin의 od 로 출력한 화면

TOY 시뮬레이터 toysim의 실행

prog3-6.o 의 실행

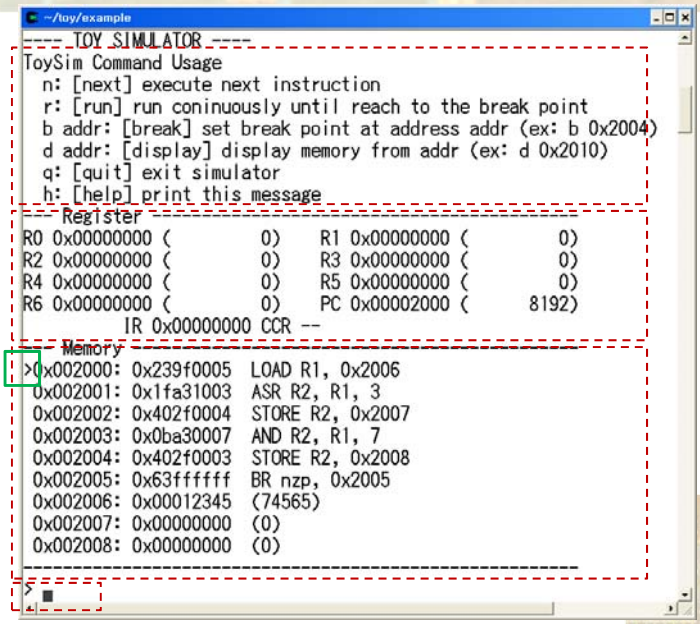
명령창에서 파일이름 입력

```
~/toy
%
%
% ./toysim prog3-6
```

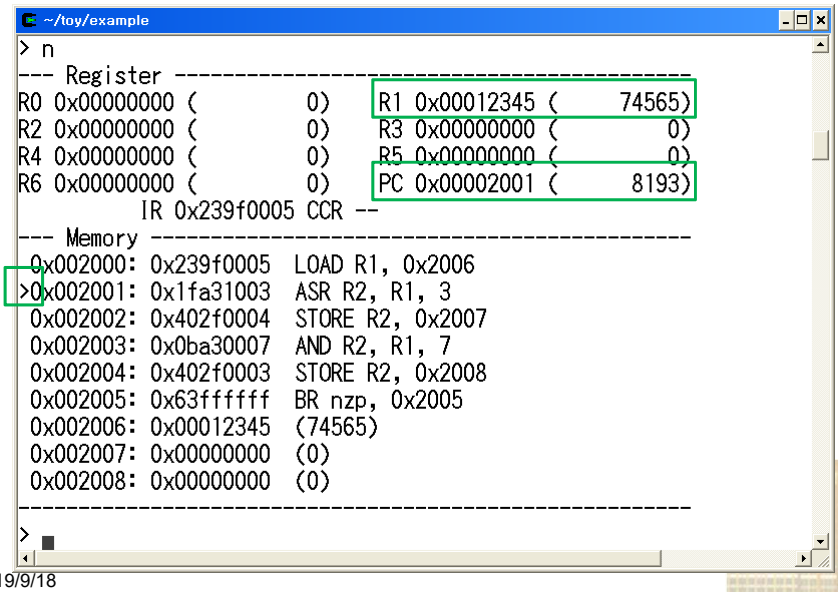
실행화면에서 파일이름 입력

```
C:₩cygwin₩home₩김용석₩toy₩toysim.exe
file name: prog3-6
```

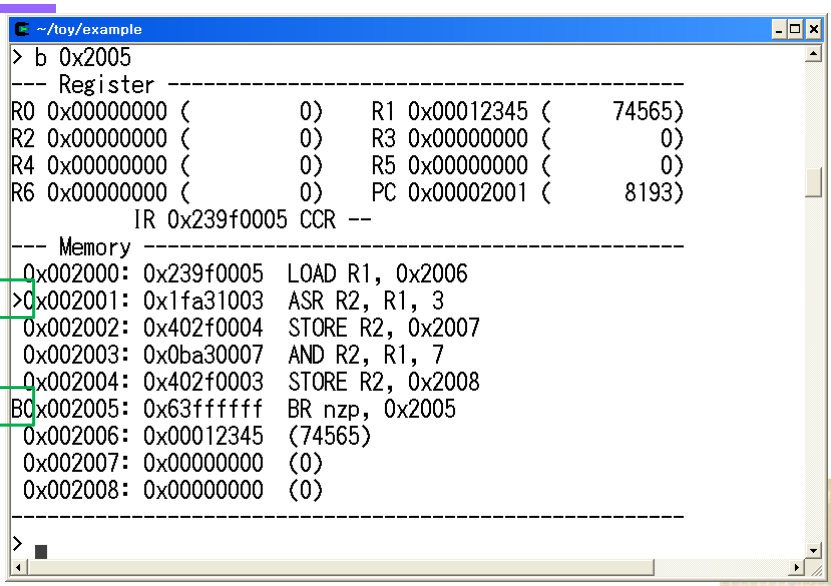
Toysim 실행 시작화면



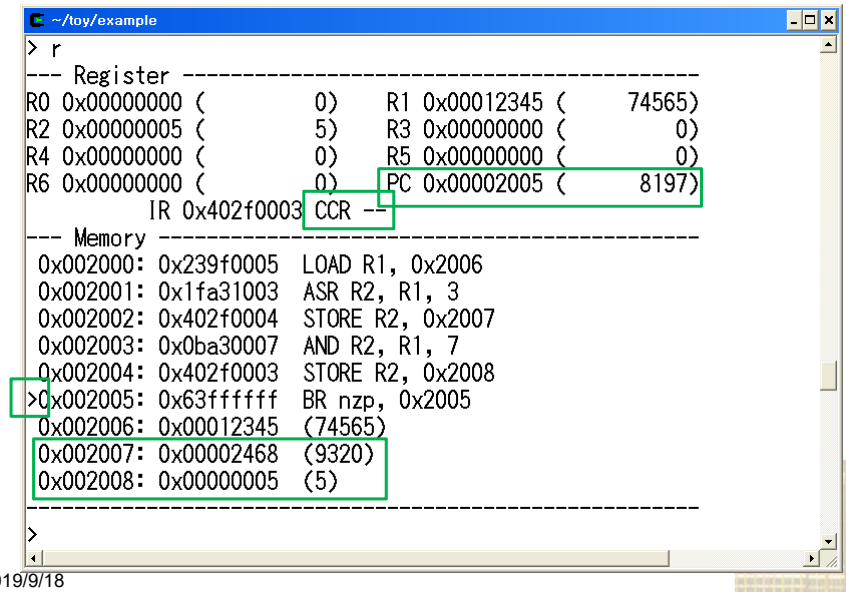
0x2000 번지의 명령 실행 후



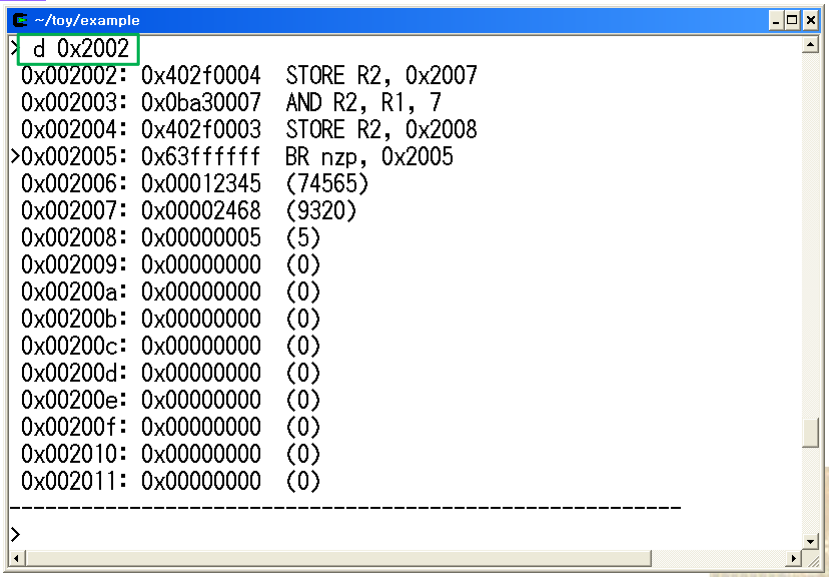
0x2005 번지에 정지점 설정 후



정지점 0x2005 번지까지 실행한 후



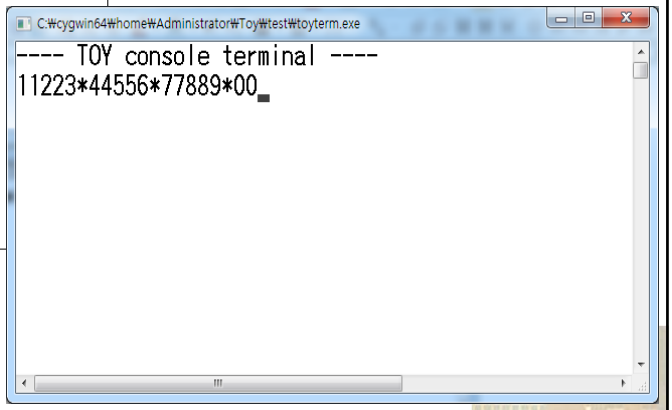
0x2002 번지부터의 메모리 내용 표시



터미널 에뮬레이터의 실행

- 369 프로그램의 toyterm 실행화면 (에코가 있음)

```
    .ORIGIN 0x2000
loop: SWI 0
    CMP R0, '3'
    BR z, match
    CMP R0, '6'
    BR z, match
    CMP R0, '9'
    BR np, print
match: COPY R0, '*'
print: SWI 1
    BR nzp, loop
```



에코가 없는 369 프로그램

```
    .ORIGIN 0x2000
    COPY R0, 0          ; 에코 없음에 해당하는 값
    SWI 4               ; 에코 설정 요청
loop: SWI 0
    CMP R0, '3'
    BR z, match
    CMP R0, '6'
    BR z, match
    CMP R0, '9'
    BR np, print
match: COPY R0, '*'
print: SWI 1
    BR nzp, loop
```