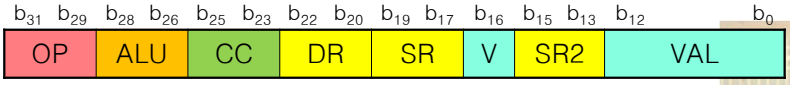
제10장
TOY 프로세서 설계

Youtube 주소
[10-1] http://youtu.be/1TjF8ed6tTI
[10-2] http://youtu.be/CXmdSvKee0o
[10-3] http://youtu.be/KX428TDMYVc
[10-4] http://youtu.be/HNHHqo0C3Mw
[10-3(추가)] http://youtu.be/sHcata0XrtE

단순한 회로를 위한 설계 (1) – 일관된 형식

- **회로의 단순화를 위해 IR의 각 부분 코드 값들을 그대로 회로의 각 부분에 전달하도록 기계어 형식 결정 (명령어 종류에 상관없이 동일하게)**
- OP: 명령어 그룹 구분
- ALU: OP별 세부 명령어 구분 (ALU에 직접 전달되는 3비트의 기능선택)
- DR: 결과저장 대상 레지스터 표시 (STORE/STR 제외)
- SR: ALU의 1번 오퍼란드 레지스터 표시 (COPY 제외)
- V: ALU의 2번 오퍼란드 종류 구별 (1: 즉석값, 0:레지스터)
  - ✓ SR2: 2번 오퍼란드가 레지스터일 경우 그 번호
  - ✓ SR2/VAL(16비트): 2번 오퍼란드가 즉석 값일 경우 그 값
- CC: 결과 데이터를 DR 레지스터에 기록할 조건
  - ✓ IR의 CC 부분의 NZP 와 CCR의 NZ 비트들의 조합으로 결정



㈜ *SWI와 RTI는 TOY 설계에서 제외함*

예외적인 기계어 형식

- STORE/STR
  - ✓값을 기록할 목적지 레지스터는 필요가 없음
  - ✓메모리에 전달할 소스 레지스터, 주소를 지정할 레지스터가 필요
  - ✓주소를 지정하는 방법은 LOAD/LDR과 동일한 방법 적용
    - → DR 부분을 소스 레지스터 표현으로 활용
    - → 회로에서 이 부분에 대한 예외처리 필요함
- COPY
  - ✓소스 오퍼랜드가 1개만 필요함
  - ✓레지스터일 수도 있고, 즉석 값일 수도 있어야 함
    - → 1번 오퍼랜드 부분은 무시하고, 2번 오퍼랜드 부분의 SR2/VAL 부분을 사용함

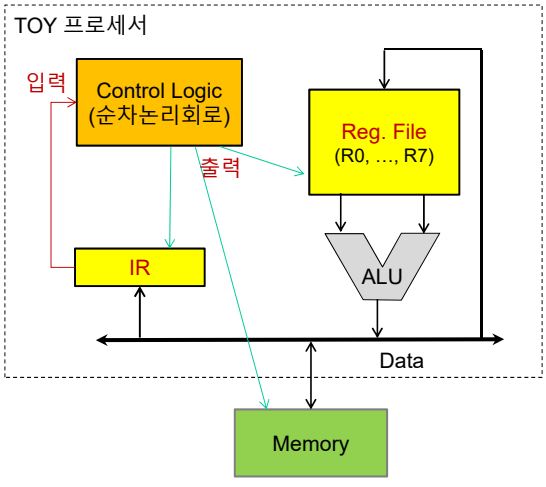
TOY 프로세서 일관된 기계어 형식

	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
LSR	0 0 0	1 1 0	1 1 1	Rd	Rs	1	0 0 0	1
ASL	0 0 0	1 1 1	1 1 1	Rd	Rs	1	0 0 0	0
ASR	0 0 0	1 1 1	1 1 1	Rd	Rs	1	0 0 0	1
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	
RTI	1 0 0	0 0 0	1 1 1	1 1 1	1 1 0	1	0 0 0	0 0 ..... 0 0

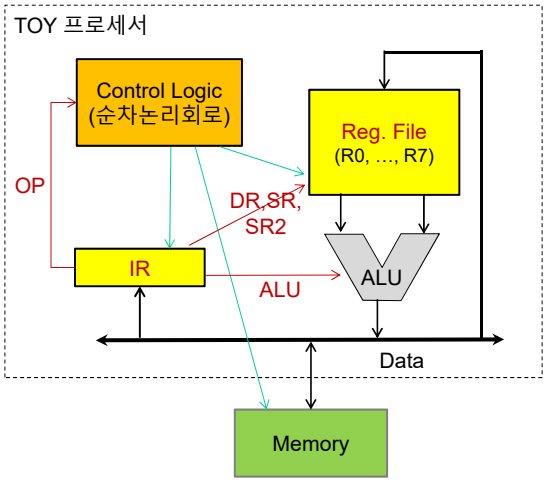
CC 코드로 DR 저장여부 선택

- IR의 CC 부분의 NZP 와 CCR의 NZ 비트들의 조합으로 결정
  - ✓OR(AND(CC[N], CCR[N]), AND(CC[Z], CCR[Z]), AND(CC[P], NOT(CCR[N]), NOT(CCR[Z]))
- CMP: SUB 와 동일하나 DR 에 저장하지 않음
  - ✓CC = 000 (SUB는 CC가 111)
  - ✓DR 은 무의미함 → 000 으로 처리
- STORE/STR: 결과를 레지스터에 저장하지 않음
  - ✓CC = 000
- BR/BRR: 조건에 따라 분기 (PC 값 변경)
  - ✓CC의 NZP 를 필요한대로 설정
- 그 외 모든 명령어: 무조건 결과를 DR에 저장
  - ✓CC = 111

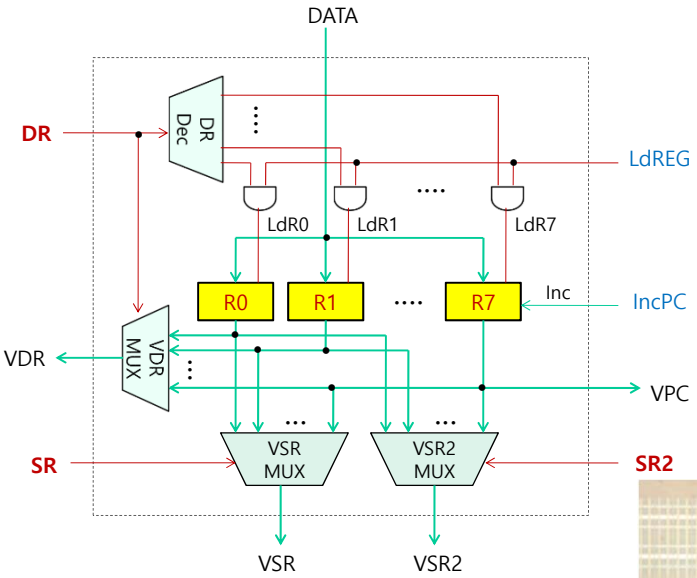
TOY 프로세서 기본 구조



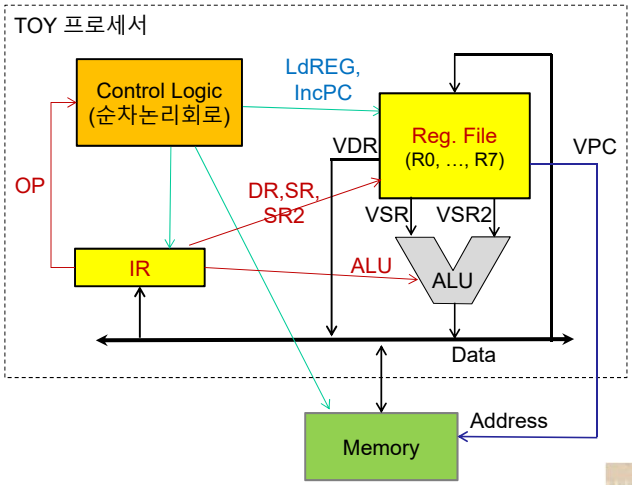
IR의 각 비트들의 연결



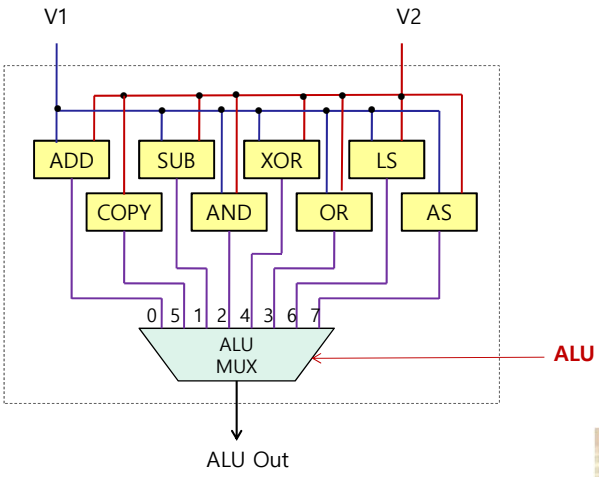
Register File의 내부 구성



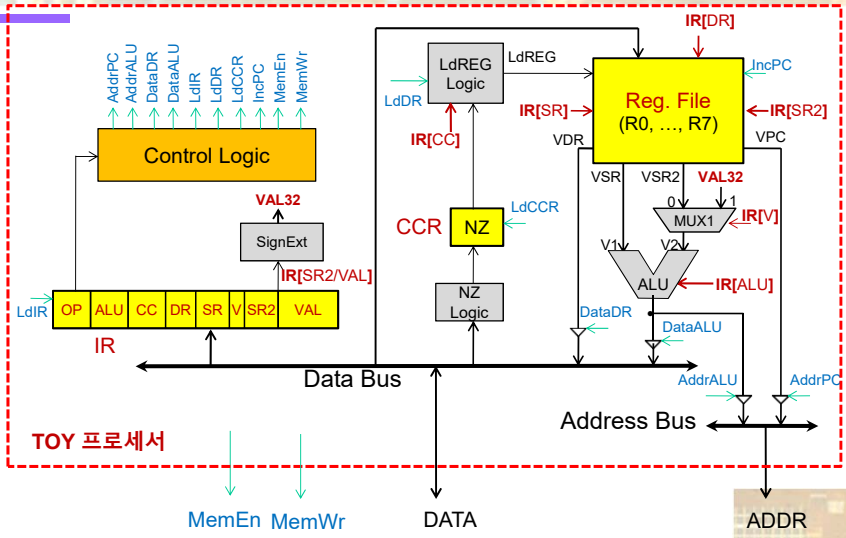
IR의 각 비트들의 연결



ALU의 내부 구성

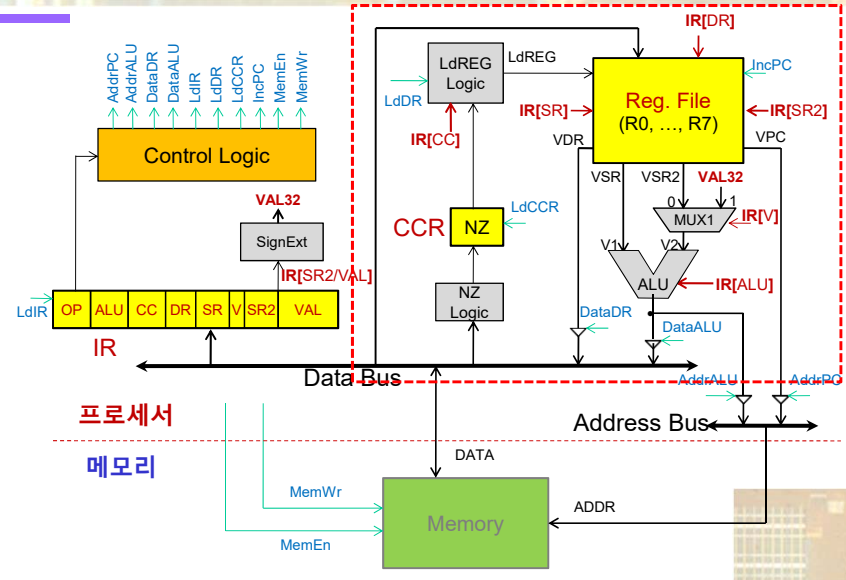


TOY 프로세서 회로도



2019/11/14 ㈜ 표기: IR[X]는 IR의 X 부분에 해당하는 코드 10-11

TOY 프로세서 전체 회로도



2019/11/14 ㈜ 표기: IR[X]는 IR의 X 부분에 해당하는 코드 10-12

단순한 회로를 위한 설계 (2) – 상태 개수 줄이기

- **유사 명령어들의 통합**
- 어셈블리 언어로는 여러 가지지만 프로세서에는 몇 가지의 기본명령어들로 처리

→21개의 명령어들을 4개의 명령어 그룹으로 처리

- ✓ALU 명령어 그룹 (ADD, SUB, … 등 12개)
- ✓LDR 명령어 그룹 (LDR, LOAD)
- ✓STR 명령어 그룹 (STR, STORE)
- ✓BRR 명령어 그룹 (BRR, BR, LEA, LINK, RET)
- ✓(SWI와 RTI는 설계에서 제외)

- 명령어 그룹은 OP 부분의 코드로 구별

기본명령어의 특수한 경우

- 기본 명령어의 특수한 경우로 처리
- NOT Rd, Rs ➔ XOR Rd, Rs, -1
- LOAD Rd, address ➔ LDR Rd, R7, offset
  - ✓LOAD의 실행: DR ← M[PC + offset]
  - ✓LDR의 실행: DR ← M[Ra + offset]
  - → LOAD는 LDR에서 Ra 부분에 PC(R7)를 적용한 것임
- STORE Rs, address ➔ STR Rs, R7, offset
  - ✓STORE의 실행: M[PC+offset] ← Rs
  - ✓STR의 실행: M[Ra+offset] ← Rs
  - →STORE는 STR에서 Ra 부분에 PC(R7)를 적용한 것임
- BR cc, address ➔ BRR cc, R7, offset
  - ✓BR의 실행: PC ← PC+offset
  - ✓BRR의 실행: PC ← Ra+offset
  - → BR은 BRR에서 Ra 부분에 PC(R7)를 적용한 것임

ALU 명령어 그룹

- ALU 기능 선택만 다를 뿐 실행동작은 동일함

➔ 동일한 OP 코드 (000) 부여

명령어	TOY 프로세서 동작
ADD	Rd ← Rs + Rs2
SUB(CMP)	Rd ← Rs - Rs2
AND	Rd ← Rs AND Rs2
OR	Rd ← Rs OR Rs2
XOR	Rd ← Rs XOR Rs2
COPY	Rd ← Rs2
LSL/LSR	Rd ← Rs LSL/LSR Rs2
ASL/ASR	Rd ← Rs ASL/ASR Rs2

ALU 명령어 그룹의 기계어 코드

	b_{31} b_{29}	b_{28} b_{26}	b_{25} b_{23}	B_{22} b_{20}	b_{19} b_{17}	b_{16}	b_{15} b_{13}	b_{12} b_0
명령어	OP	ALU	CC	DR	SR	V	SR2	VAL
ADD	000	000	111	Rd	Rs			
SUB		001	111	Rd	Rs			
CMP		001	000	Rd	Rs			
AND		010	111	Rd	Rs			
OR		011	111	Rd	Rs			
XOR		100	111	Rd	Rs			
COPY		101	111	Rd	Rs			
LSL/LSR		110	111	Rd	Rs			
ASL/ASR		111	111	Rd	Rs			

조건부 실행 명령어를 추가하려면?

LDR 명령어 그룹

LDR 명령어 그룹의 동작

명령어	TOY 프로세서 동작
LOAD	Rd ← M[R7 + offset]
LDR	Rd ← M[Ra + offset]

- 동일한 OP 코드 (001) 부여
- ALU=000 ← Ra + offset 동작을 ALU의 ADD 기능 활용

	b_{31} b_{29}	b_{28} b_{26}	b_{25} b_{23}	b_{22} b_{20}	b_{19} b_{17}	b_{16}	b_{15} b_{13}	b_{12} b_0
명령어	OP	ALU	CC	DR	SR	V	SR2	VAL
LOAD	0 0 1	0 0 0	1 1 1	Rd	R7	1	offset	
LDR		0 0 0	1 1 1	Rd	Ra	1	offset	

STR 명령어 그룹

STR 명령어 그룹의 동작

명령어	TOY 프로세서 동작
STORE	M[R7 + offset] ← Rs
STR	M[Ra + offset] ← Rs

- 동일한 OP 코드 (010) 부여
- ALU=000 ← Ra + offset 동작을 ALU의 ADD 기능 활용

	b_{31} b_{29}	b_{28} b_{26}	b_{25} b_{23}	b_{22} b_{20}	b_{19} b_{17}	b_{16}	b_{15} b_{13}	b_{12} b_0
명령어	OP	ALU	CC	DR	SR	V	SR2	VAL
STORE	0 1 0	0 0 0	0 0 0	Rs	R7	1	offset	
STR		0 0 0	0 0 0	Rs	Ra	1	offset	

BRR 명령어 그룹

- 명령어의 실행
  - ✓BR 의 실행: 조건 만족시 R7 ← R7 + offset
  - ✓BRR 의 실행: 조건 만족시 R7 ← Ra + offset
  - ✓LEA 의 실행: Rd ← R7 + offset
  - ✓LINK 의 실행: R6 ← R7 + 1
  - ✓RET 의 실행: R7 ← R6 + 0
- 모두 동일한 실행 형식: DR ← SR + offset
- CCR은 갱신하지 않음

(주) ADD나 COPY 명령어를 사용하면 CCR 이 갱신됨

BRR 명령어 그룹의 기계어 코드

- 동일한 OP 코드 (011) 부여
- ALU=000 ← SR + offset 동작을 ALU의 ADD 기능 활용

	b_{31} b_{29}	b_{28} b_{26}	b_{25} b_{23}	b_{22} b_{20}	b_{19} b_{17}	b_{16}	b_{15} b_{13}	b_{12} b_0
명령어	OP	ALU	CC	DR	SR	V	SR2	VAL
BR	0 1 1	0 0 0	N Z P	R7	R7	1	offset	
BRR		0 0 0	N Z P	R7	Ra	1	offset	
LEA		0 0 0	1 1 1	Rd	R7	1	offset	
LINK		0 0 0	1 1 1	R6	R7	1	1	
RET		0 0 0	1 1 1	R7	R6	1	0	

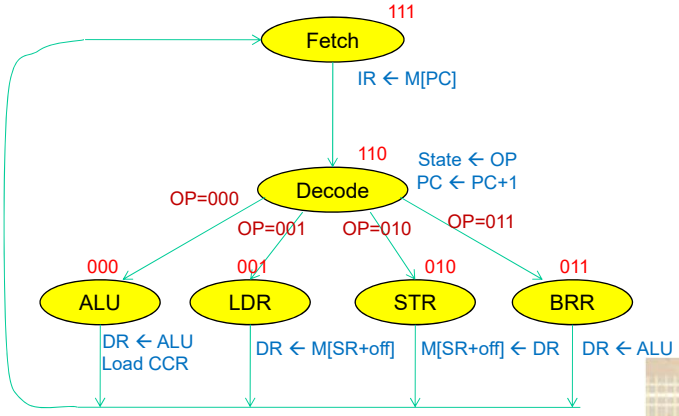
TOY 프로세서 기계어 코드

- 4가지 명령어 그룹 (OP 코드로 구별)

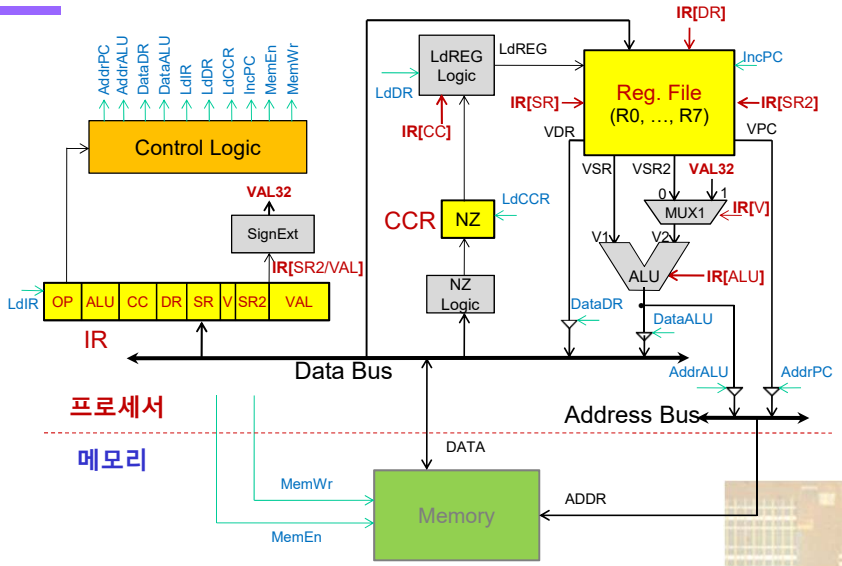
	b_{31} b_{29}	b_{28} b_{26}	b_{25} b_{23}	b_{22} b_{20}	b_{19} b_{17}	b_{16}	b_{15} b_{13}	b_{12} b_0
그룹	OP	ALU	CC	DR	SR	V	SR2	VAL
ALU	0 0 0	ALU	N Z P	Rd	Rs	V		
LDR	0 0 1	0 0 0	1 1 1	Rd	Ra	1	offset	
STR	0 1 0	0 0 0	0 0 0	Rs	Ra	1	offset	
BRR	0 1 1	0 0 0	N Z P	Rd	Ra	1	offset	

Control Logic 상태 천이도

- 상태번호는 OP 코드와 일치하도록
- ← Decode 처리를 간단하게 하기 위함

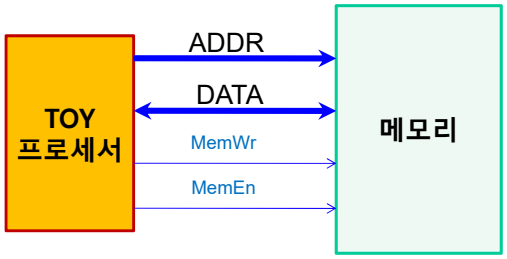


TOY 프로세서 전체 회로도



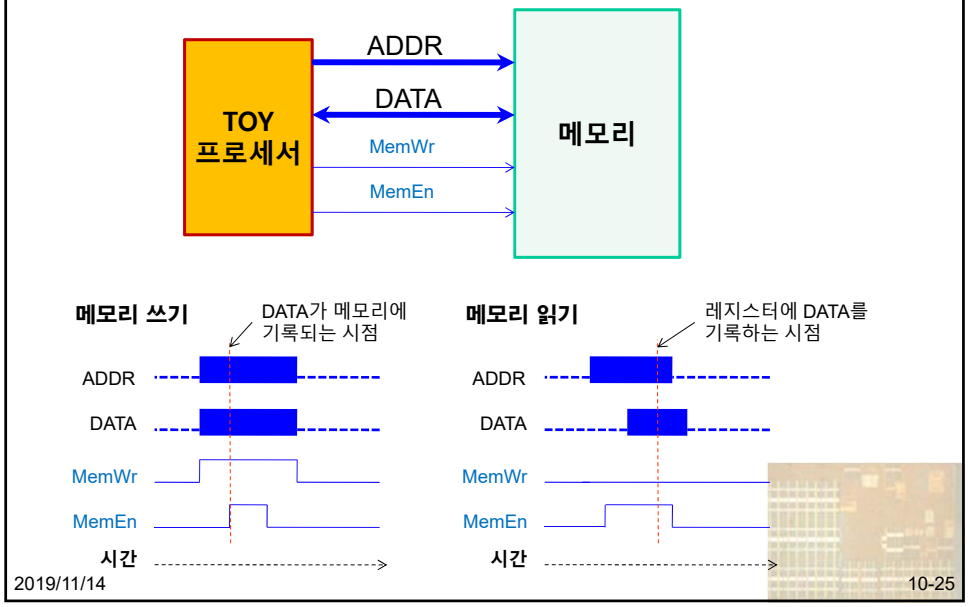
2019/11/14 ※ 표기: IR[X]는 IR의 X 부분에 해당하는 코드 10-23

메모리의 읽기와 쓰기를 위한 신호

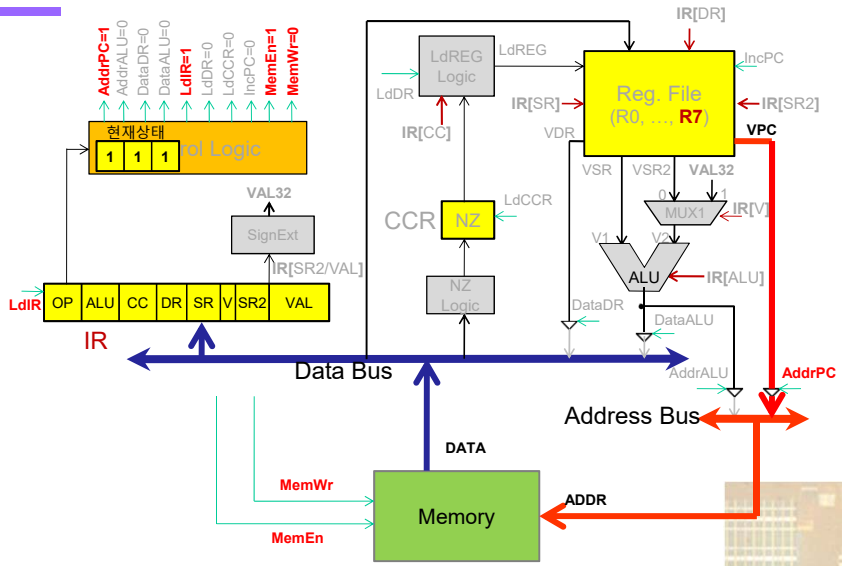


2019/11/14 10-24

(참고) 메모리 읽기/쓰기 신호 타이밍 차트



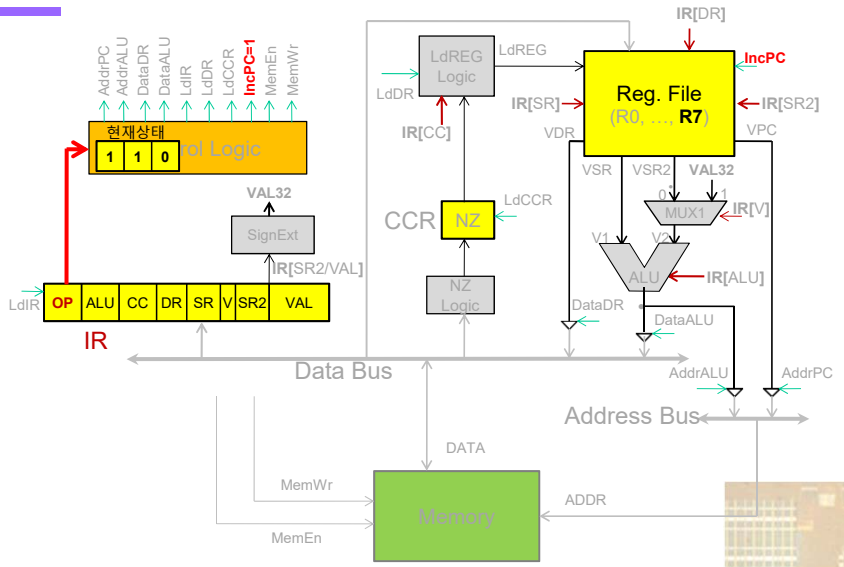
상태 111: 명령어 읽기



Control Logic 신호 진리표

입력		출력										
현재상태 $S_2S_1S_0$	OP	Next $N_2N_1N_0$	AddrPC	AddrALU	DataDR	DataALU	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
0 0 0												
0 0 1												
0 1 0												
0 1 1												
1 0 0												
1 0 1												
1 1 0												
1 1 1	x x x	1 1 0	1	0	0	0	1	0	0	0	1	0

상태 110: 명령어 해독



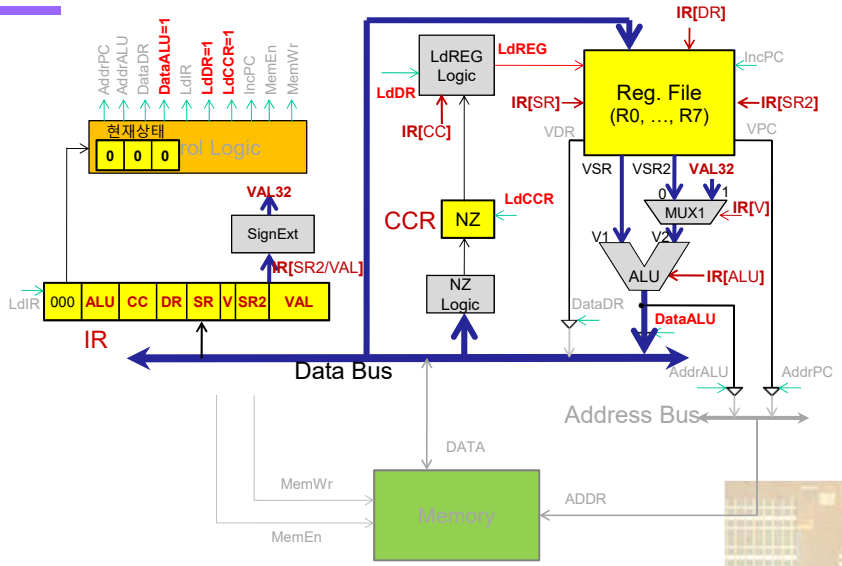
명령어 해독 상태의 제어 로직 신호

입력		출력										
현재상태 $S_2S_1S_0$	OP	Next $N_2N_1N_0$	AddrPC	AddrAL	DataDR	DataAL	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
110	000	000	0	0	0	0	0	0	0	1	0	x
	001	001										
	010	010										
	011	011										
	100	100										
	101	101										
	110	110										
	111	111										

Control Logic 신호 진리표

입력		출력										
현재상태 $S_2S_1S_0$	OP	Next $N_2N_1N_0$	AddrPC	AddrALU	DataDR	DataALU	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
000												
001												
010												
011												
100												
101												
110	OP	OP	0	0	0	0	0	0	0	1	0	x
111	x x x	110	1	0	0	0	1	0	0	0	1	0

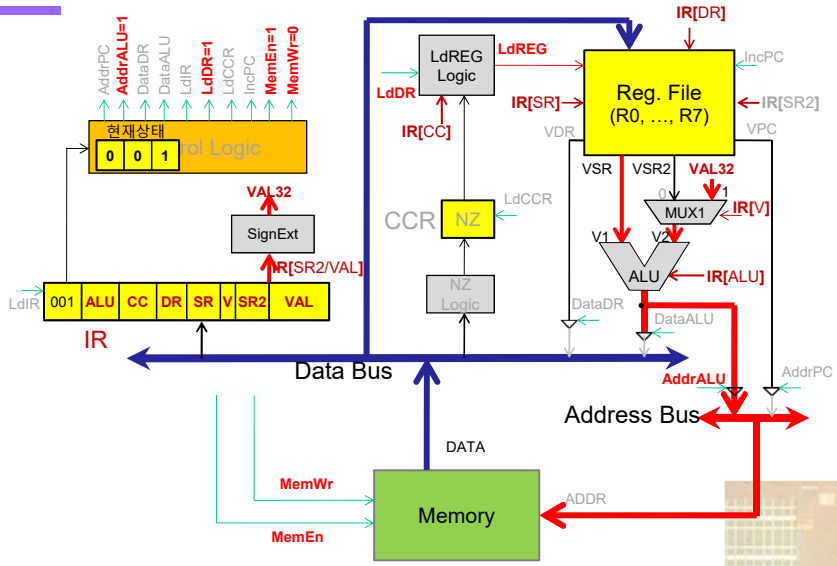
상태 000: ALU 명령어 실행



Control Logic 신호 진리표

입력		출력										
현재상태 $S_2S_1S_0$	OP	Next $N_2N_1N_0$	AddrPC	AddrALU	DataDR	DataALU	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
0 0 0	x x x	1 1 1	0	0	0	1	0	1	1	0	0	x
0 0 1												
0 1 0												
0 1 1												
1 0 0												
1 0 1												
1 1 0	OP	OP	0	0	0	0	0	0	0	1	0	x
1 1 1	x x x	1 1 0	1	0	0	0	1	0	0	0	1	0

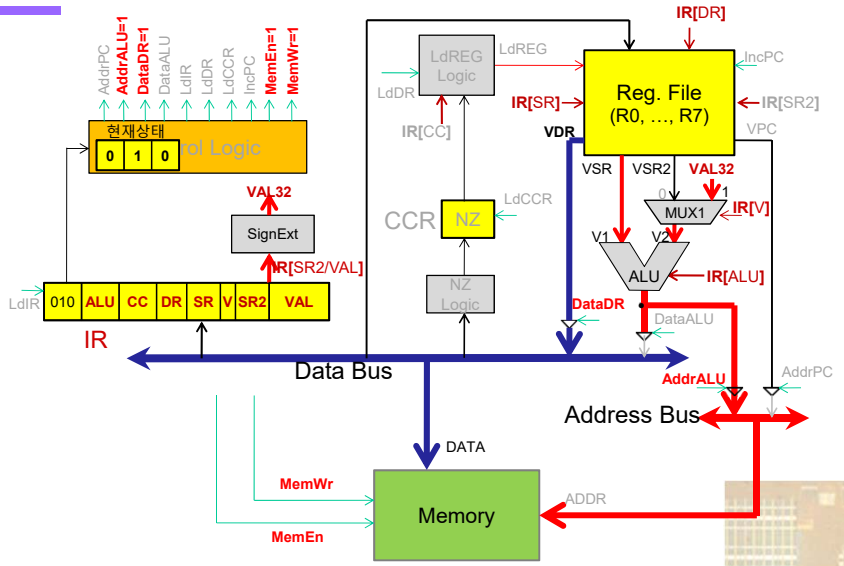
상태 001: LDR 명령어 실행



Control Logic 신호 진리표

입력		출력										
현재상태 $S_2S_1S_0$	OP	Next $N_2N_1N_0$	AddrPC	AddrALU	DataDR	DataALU	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
0 0 0	x x x	1 1 1	0	0	0	1	0	1	1	0	0	x
0 0 1	x x x	1 1 1	0	1	0	0	0	1	0	0	1	0
0 1 0												
0 1 1												
1 0 0												
1 0 1												
1 1 0	OP	OP	0	0	0	0	0	0	0	1	0	x
1 1 1	x x x	1 1 0	1	0	0	0	1	0	0	0	1	0

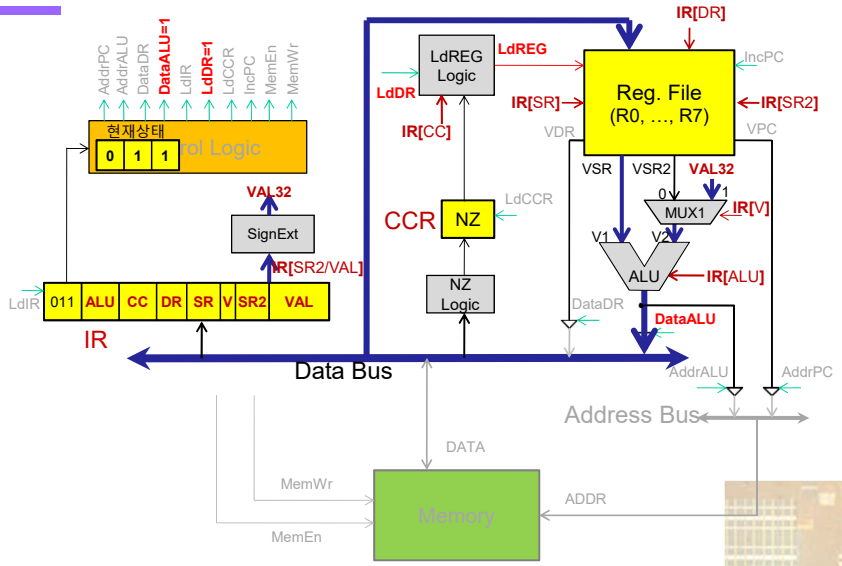
상태 010: STR 명령어 실행



Control Logic 신호 진리표

입력		출력										
현재상태 $S_2S_1S_0$	OP	Next $N_2N_1N_0$	AddrPC	AddrALU	DataDR	DataALU	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
0 0 0	x x x	1 1 1	0	0	0	1	0	1	1	0	0	x
0 0 1	x x x	1 1 1	0	1	0	0	0	1	0	0	1	0
0 1 0	x x x	1 1 1	0	1	1	0	0	0	0	0	1	1
0 1 1												
1 0 0												
1 0 1												
1 1 0	OP	OP	0	0	0	0	0	0	0	1	0	x
1 1 1	x x x	1 1 0	1	0	0	0	1	0	0	0	1	0

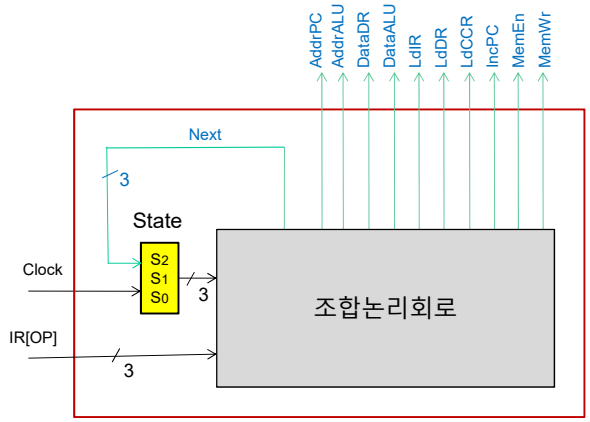
상태 011: BRR 명령어 실행



Control Logic 신호 진리표

입력		출력										
현재상태 $S_2S_1S_0$	OP	Next $N_2N_1N_0$	AddrPC	AddrALU	DataDR	DataALU	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
0 0 0	x x x	1 1 1	0	0	0	1	0	1	1	0	0	x
0 0 1	x x x	1 1 1	0	1	0	0	0	1	0	0	1	0
0 1 0	x x x	1 1 1	0	1	1	0	0	0	0	0	1	1
0 1 1	x x x	1 1 1	0	0	0	1	0	1	0	0	0	x
1 0 0												
1 0 1												
1 1 0	OP	OP	0	0	0	0	0	0	0	1	0	x
1 1 1	x x x	1 1 0	1	0	0	0	1	0	0	0	1	0

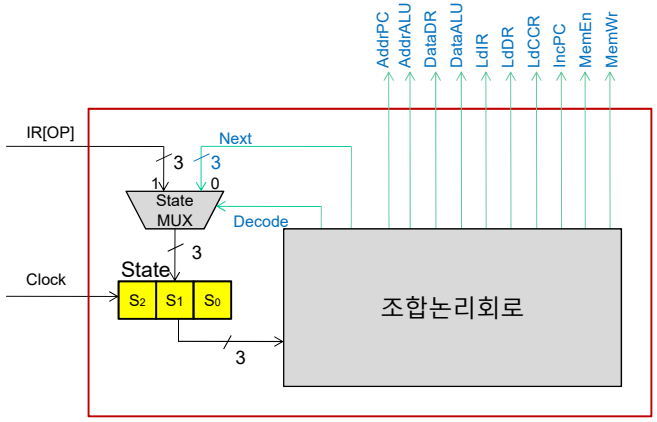
Control Logic 순차논리회로 구성



상태 플립플롭의 다음 상태 처리

- 조합논리회로 부분의 입력이 6비트인데 OP 부분을 제외하여 3비트로 줄이면 회로가 훨씬 단순해질 것임
- 동작 특성 분석
  - ✓명령어 해독 (110) 상태에서는 외부 입력인 OP 3비트를 저장
  - ✓이외의 상태에서는 지정된 Next 3비트를 저장
- 처리방안: 상태 플립플롭에 OP 3비트 또는 조합논리회로에서 나오는 Next 3비트 를 선택하여 저장
  - ✓멀티플렉서를 활용하여 명령어 해독 상태일 때는 OP 3비트를 선택, 이외의 상태일 때는 Next 3비트를 선택
  - ✓멀티플렉서 선택신호로서 명령어 해독 상태를 표시하는 신호(Decode) 필요
  - ✓조합논리회로 입력에는 OP 3비트가 필요 없음

Control Logic 순차논리회로 구성



Control Logic 신호 진리표

현재상태	Decode	Next	AddrPC	AddrALU	DataDR	DataALU	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
000	0	111	0	0	0	1	0	1	1	0	0	x
001	0	111	0	1	0	0	0	1	0	0	1	0
010	0	111	0	1	1	0	0	0	0	0	1	1
011	0	111	0	0	0	1	0	1	0	0	0	x
100	0											
101	0											
110	1	xxx	0	0	0	0	0	0	0	1	0	x
111	0	110	1	0	0	0	1	0	0	0	1	0

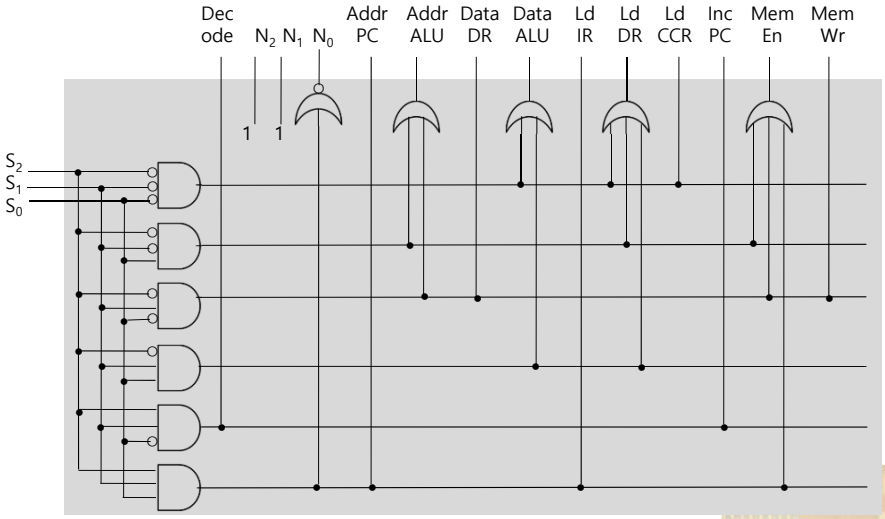
Control Logic 신호 진리표 (상태 100, 101 보완)

현재상태	Decode	Next	AddrPC	AddrALU	DataDR	DataALU	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
000	0	111	0	0	0	1	0	1	1	0	0	x
001	0	111	0	1	0	0	0	1	0	0	1	0
010	0	111	0	1	1	0	0	0	0	0	1	1
011	0	111	0	0	0	1	0	1	0	0	0	x
100	0	111	0	0	0	0	0	0	0	0	0	x
101	0	111	0	0	0	0	0	0	0	0	0	x
110	1	xxx	0	0	0	0	0	0	0	1	0	x
111	0	110	1	0	0	0	1	0	0	0	1	0

Control Logic 신호 진리표 (완성)

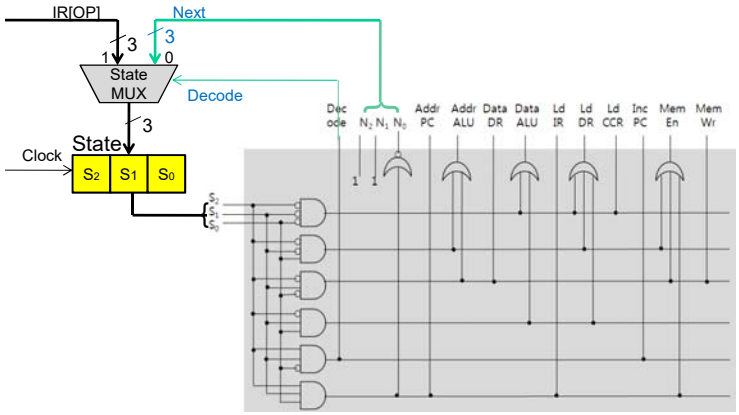
현재상태	Decode	Next	AddrPC	AddrALU	DataDR	DataALU	LdIR	LdDR	LdCCR	IncPC	MemEn	MemWr
000	0	111	0	0	0	1	0	1	1	0	0	0
001	0	111	0	1	0	0	0	1	0	0	1	0
010	0	111	0	1	1	0	0	0	0	0	1	1
011	0	111	0	0	0	1	0	1	0	0	0	0
100	0	111	0	0	0	0	0	0	0	0	0	0
101	0	111	0	0	0	0	0	0	0	0	0	0
110	1	111	0	0	0	0	0	0	0	1	0	0
111	0	110	1	0	0	0	1	0	0	0	1	0

Control Logic의 신호별 조합논리회로

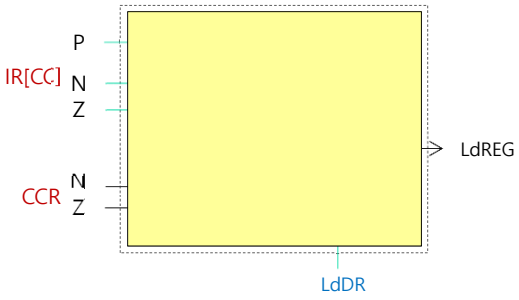


IncPC = Decode
AddrPC = LdIR
DataDR = MemWr

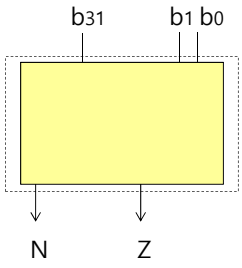
Control Logic 순차논리회로



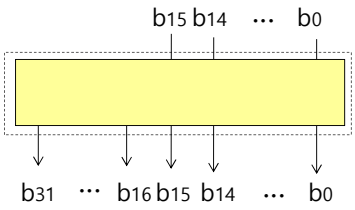
LdREG Logic의 신호 생성 회로



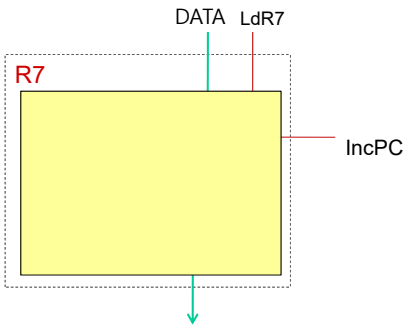
NZ Logic 의 N/Z 비트 신호 생성 회로



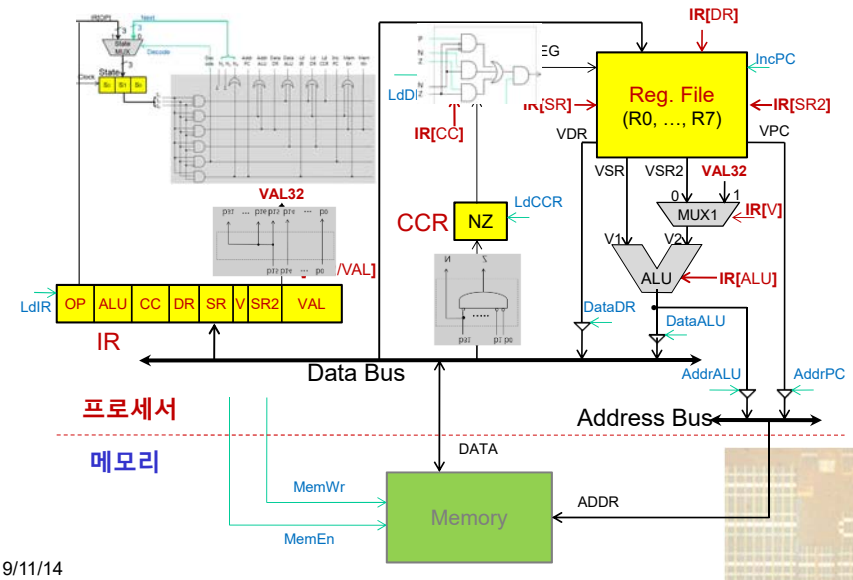
부호확장을 위한 SignExt 회로



R7의 IncPC 신호를 위한 회로



TOY 프로세서 전체 회로도 (완성)



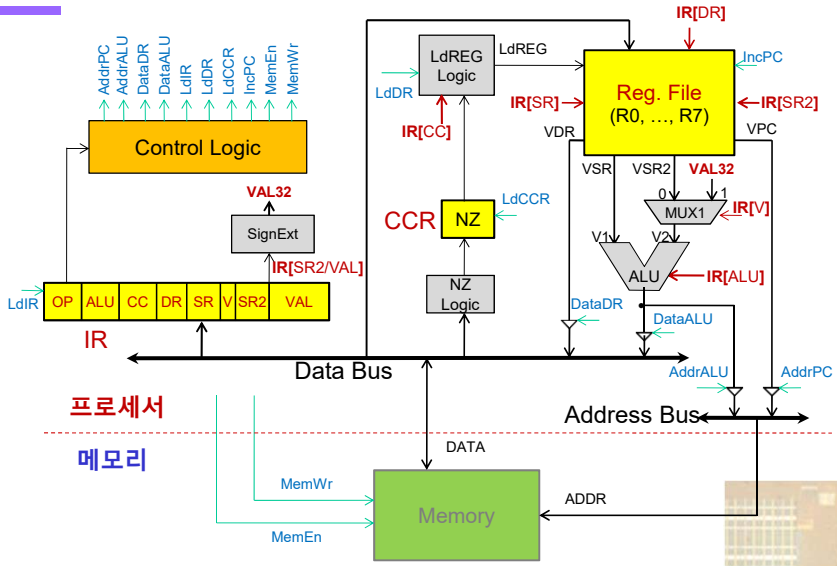
회로 각 부분의 값의 변화

x = x + y;

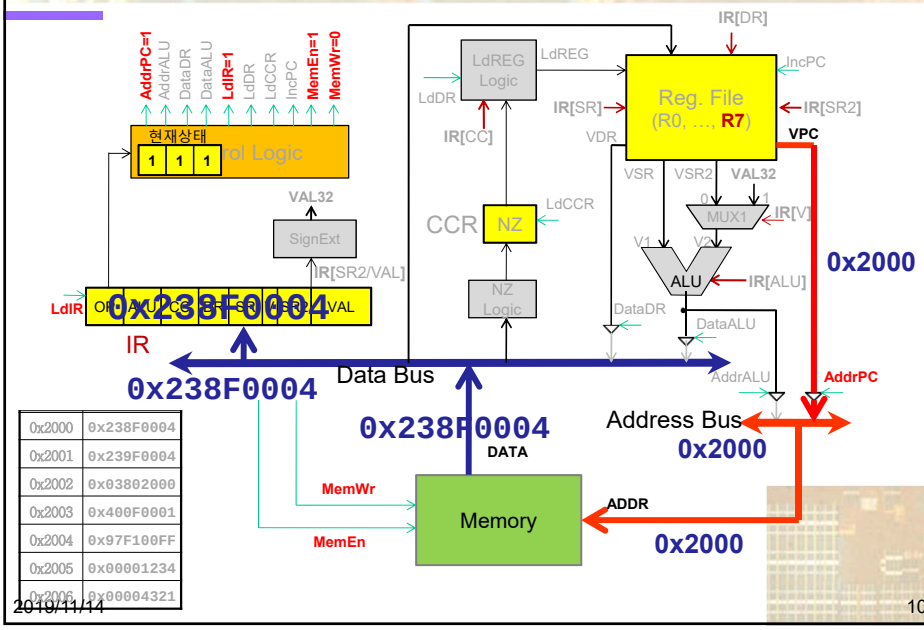
```
    .ORIGIN 0x2000
    LOAD R0, x
    LOAD R1, y
    ADD R0, R0, R1
    STORE R0, x
    SWI 255
x:  .FILL 0x1234
y:  .FILL 0x4321
```

메모리 주소	기계어코드	어셈블리 언어
0x2000	**0x238F0004**	LOAD R0, 0x2005
0x2001	**0x239F0004**	LOAD R1, 0x2006
0x2002	**0x03802000**	ADD R0, R0, R1
0x2003	**0x400F0001**	STORE R0, 0x2005
0x2004	**0x97F100FF**	SWI 255
0x2005	**0x00001234**	.FILL 0x1234
0x2006	**0x00004321**	.FILL 0x4321

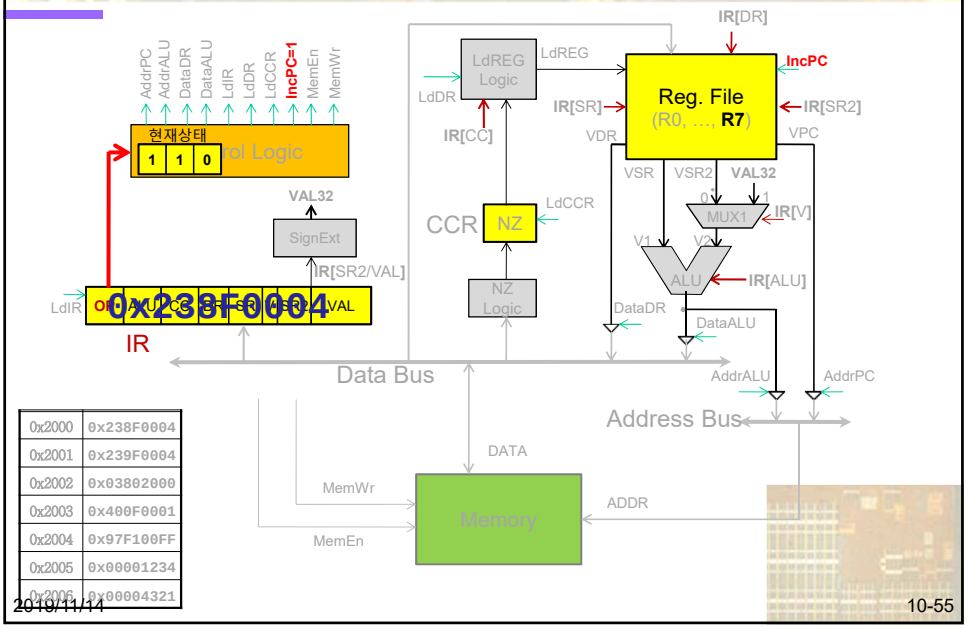
TOY 프로세서 전체 회로도



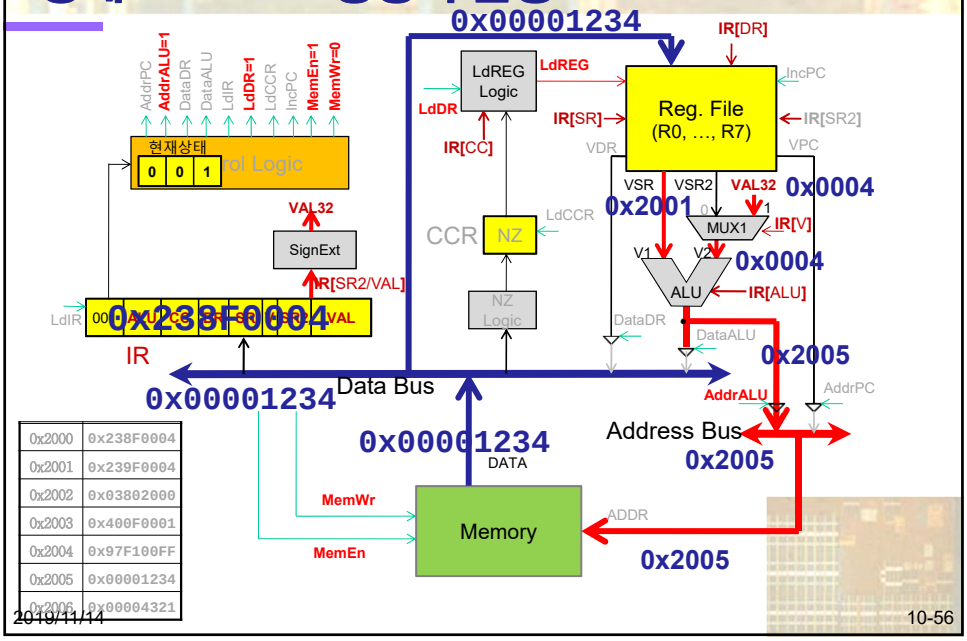
상태 111: 명령어 읽기



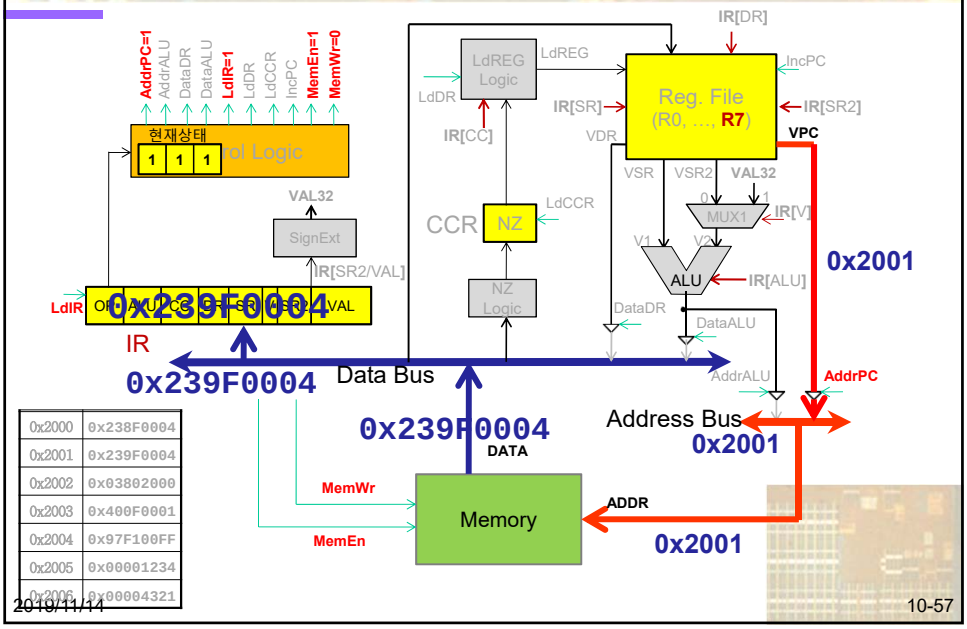
상태 110: 명령어 해독



상태 001: LDR 명령어 실행

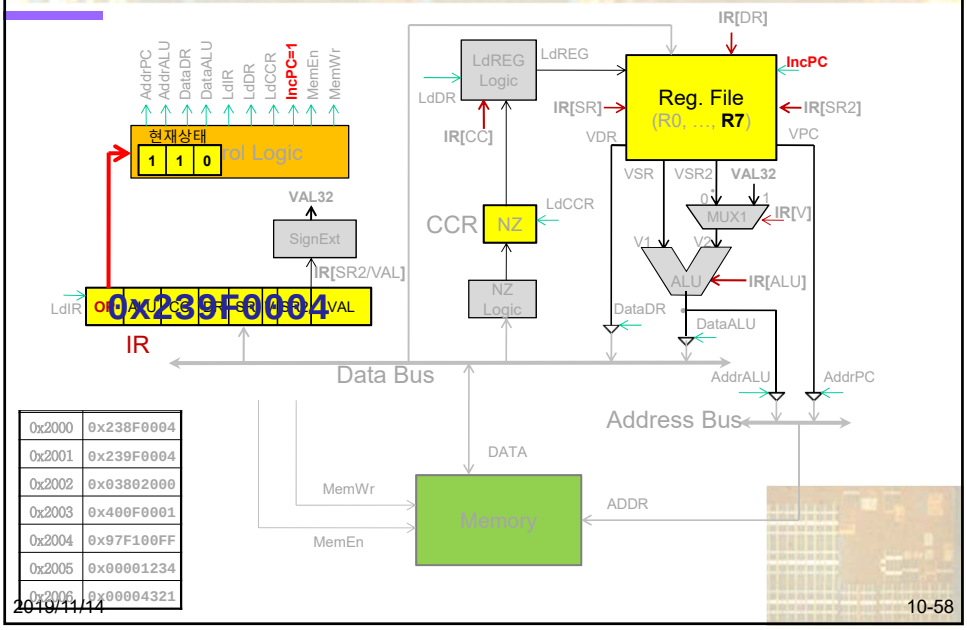


상태 111: 명령어 읽기



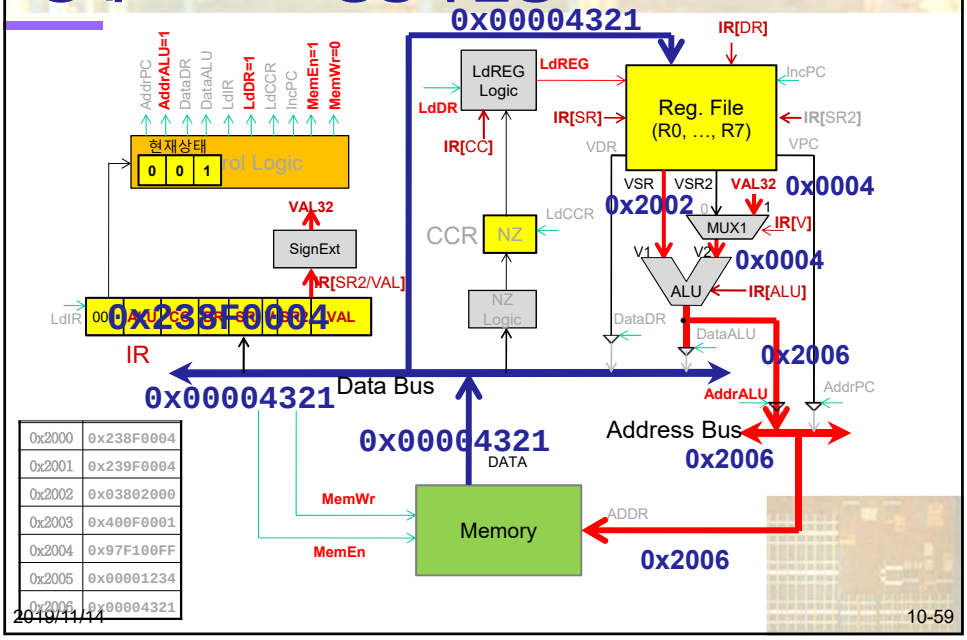
0x2000	0x238F0004
0x2001	0x239F0004
0x2002	0x03802000
0x2003	0x400F0001
0x2004	0x97F100FF
0x2005	0x00001234
0x2006	0x00004321

상태 110: 명령어 해독

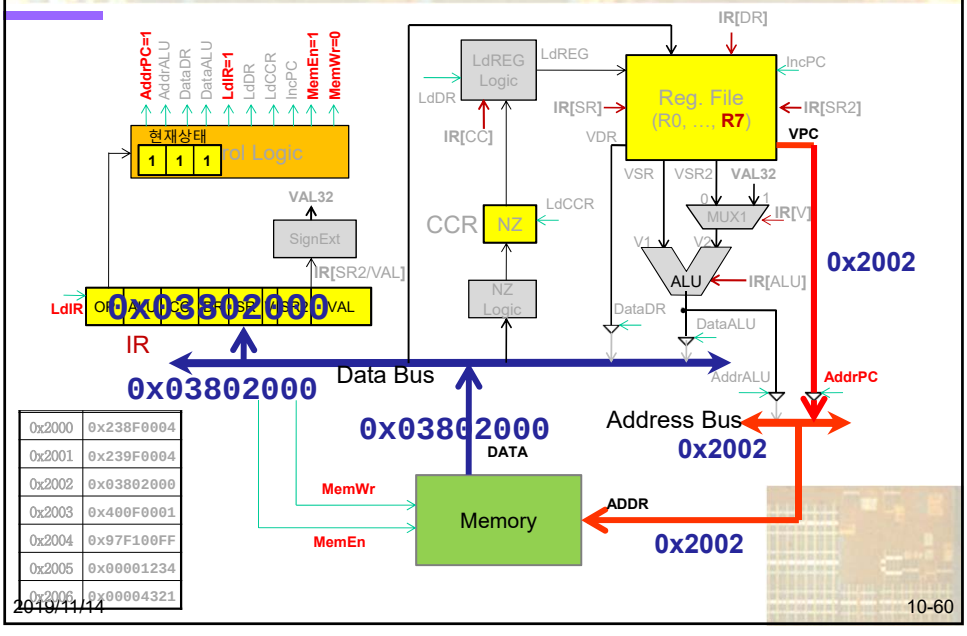


0x2000	0x238F0004
0x2001	0x239F0004
0x2002	0x03802000
0x2003	0x400F0001
0x2004	0x97F100FF
0x2005	0x00001234
0x2006	0x00004321

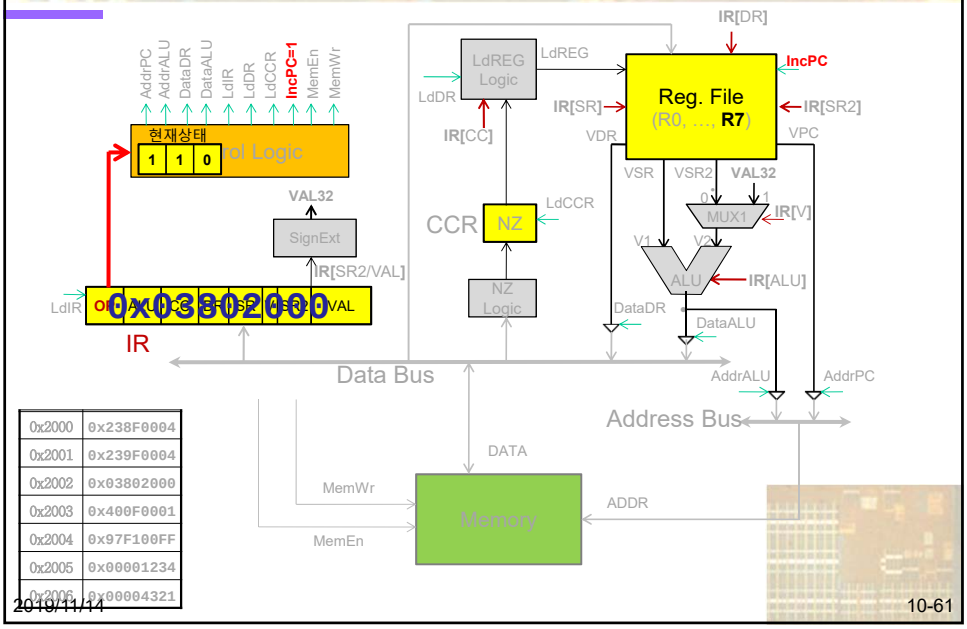
상태 001: LDR 명령어 실행



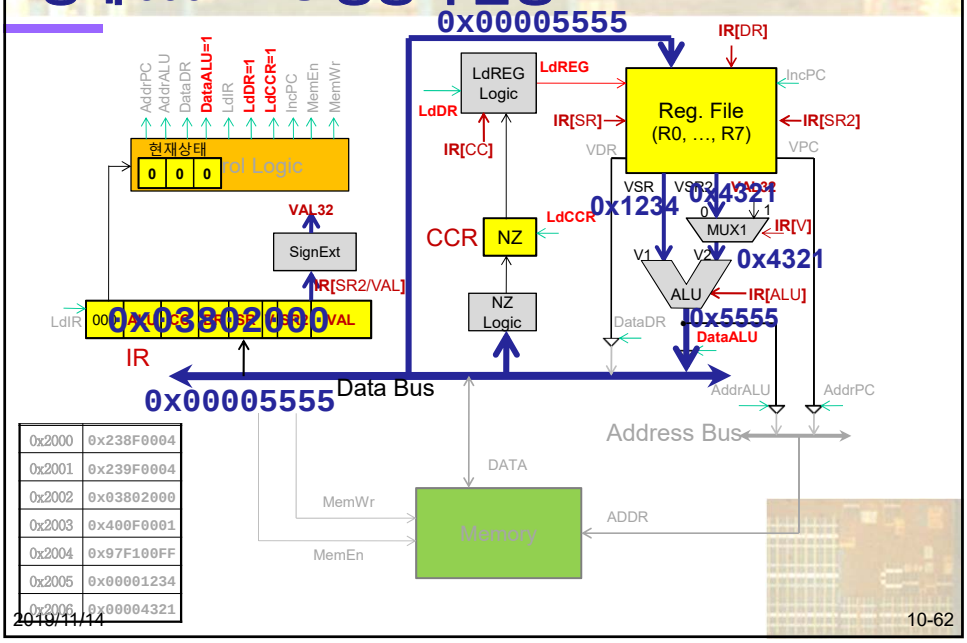
상태 111: 명령어 읽기



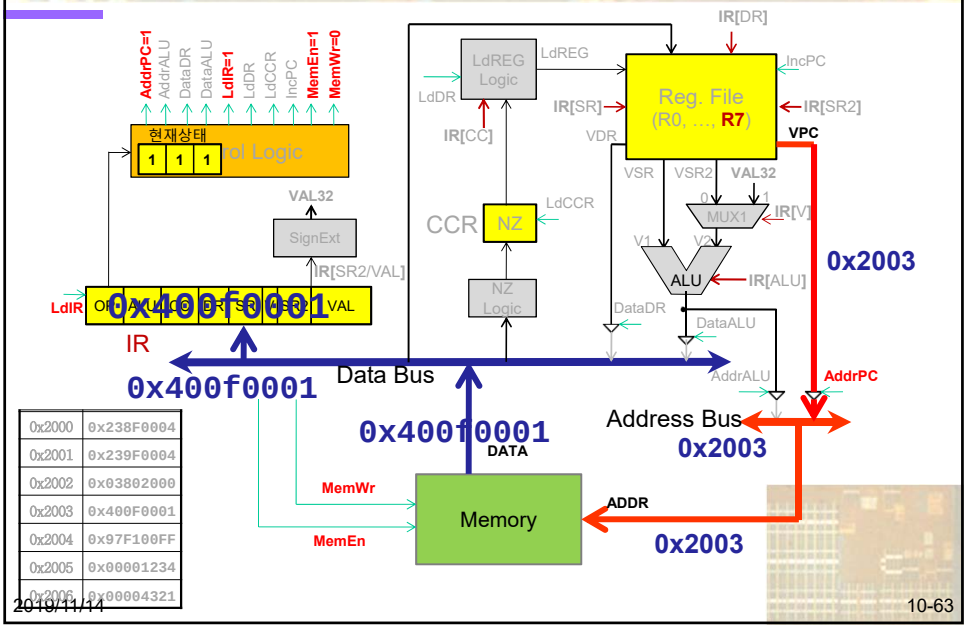
상태 110: 명령어 해독



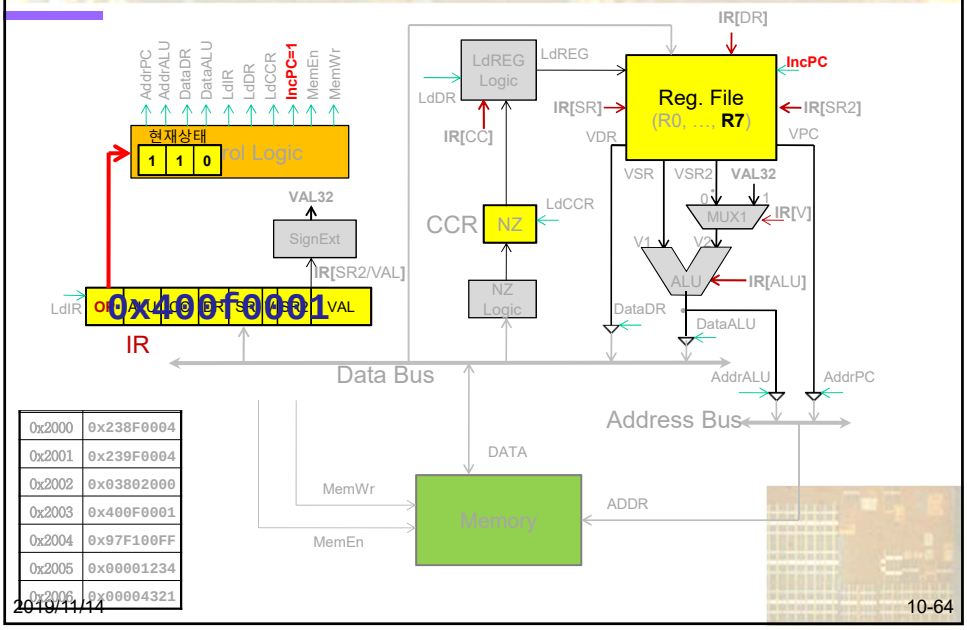
상태 000: ALU 명령어 실행



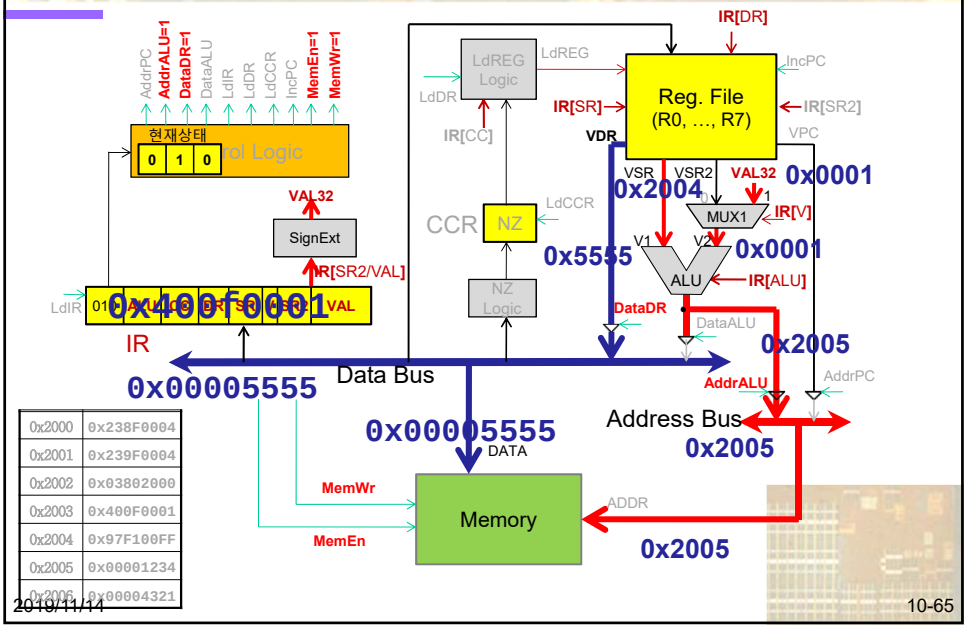
상태 111: 명령어 읽기



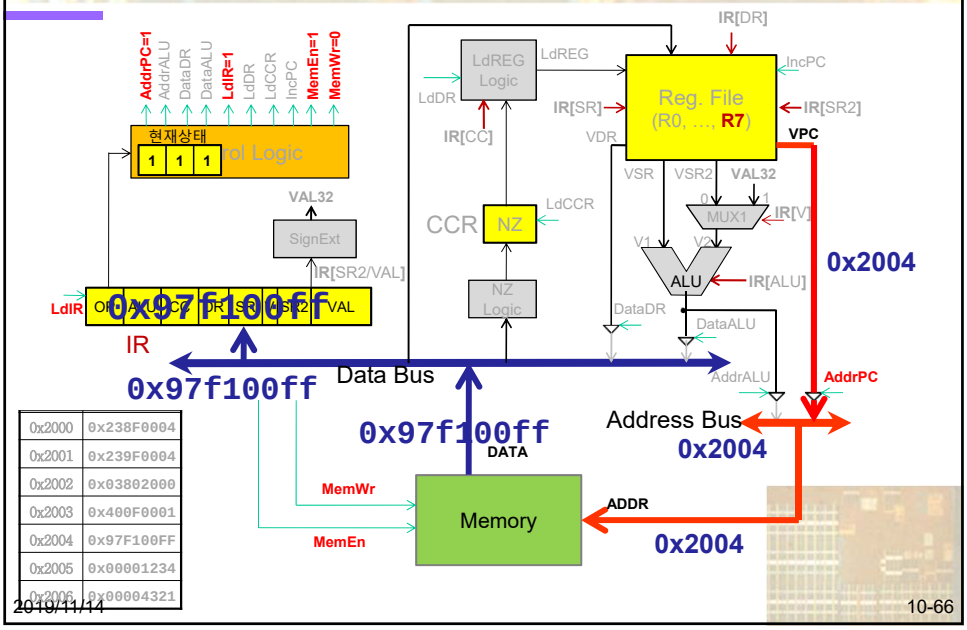
상태 110: 명령어 해독



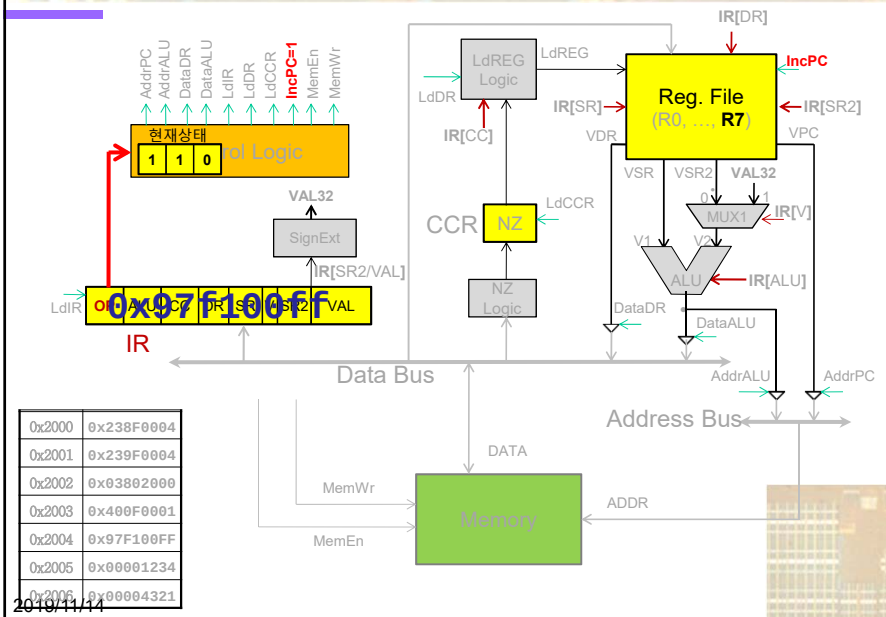
상태 010: STR 명령어 실행



상태 111: 명령어 읽기



상태 110: 명령어 해독

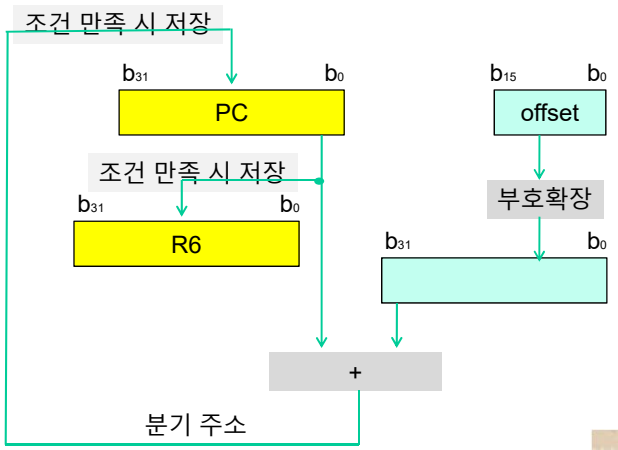


0x2000	0x238F0004
0x2001	0x239F0004
0x2002	0x03802000
0x2003	0x400F0001
0x2004	0x97F100FF
0x2005	0x00001234
0x2006	0x00004321

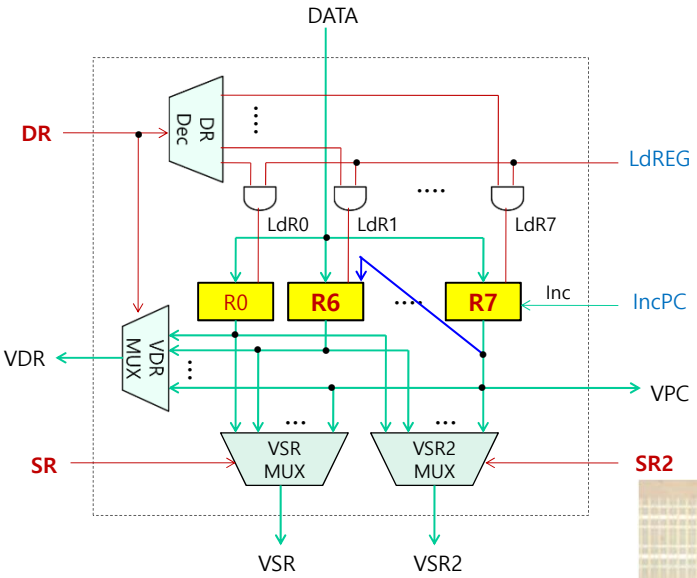
회로 각 부분의 값의 변화

클록 순서	TOY 프로세서 내부상태	PC	R0	R1	IR	주소버스의 내용	데이터버스의 내용
1	명령어 읽기	0x00002000	0x00000000	0x00000000	0x238F0004	0x00002000	0x238F0004
2	명령어 해석	0x00002001	0x00000000	0x00000000	0x238F0004	-	-
3	명령어 실행	0x00002001	0x00001234	0x00000000	0x238F0004	0x00002005	0x00001234
4	명령어 읽기	0x00002001	0x10000234	0x00000000	0x239F0004	0x00002001	0x239F0004
5	명령어 해석	0x00002002	0x00001234	0x00000000	0x239F0004	-	-
6	명령어 실행	0x00002002	0x00001234	0x00004321	0x239F0004	0x00002006	0x00004321
7	명령어 읽기	0x00002002	0x00001234	0x00004321	0x03802000	0x00002002	0x03802000
8	명령어 해석	0x00002003	0x00001234	0x00004321	0x03802000	-	-
9	명령어 실행	0x00002003	0x00005555	0x00004321	0x03802000	-	0x00005555
10	명령어 읽기	0x00002003	0x00005555	0x00004321	0x400F0001	0x00002003	0x400F0001
11	명령어 해석	0x00002004	0x00005555	0x00004321	0x400F0001	-	-
12	명령어 실행	0x00002004	0x00005555	0x00004321	0x400F0001	0x00002005	0x00005555
13	명령어 읽기	0x00002004	0x00005555	0x00004321	0x97F100FF	0x00002004	0x97F100FF
14	명령어 해석	0x00002005	0x00005555	0x00004321	0x97F100FF	-	-
15	명령어 실행	0x00002005	0x00005555	0x00004321	0x97F100FF	???	???

(참고) BRL (LINK+BR) 실행 회로 추가?



Register File의 내부 구성



SWI/RTI 실행 상태(100/101)의 회로 동작

- CCR에 모드 비트 M 추가
  - ✓0: 응용프로그램 실행 상태
  - ✓1: 운영체제 실행 상태
- 현재의 CCR 저장을 위한 레지스터 CCRs 추가
- 현재의 PC 저장을 위한 레지스터 R6s 추가
- SWI n 명령어의 실행
  - ✓CCRs ← CCR, R6s ← PC
  - ✓CCR[M] ← 1, PC ← Mem[n]
- RTI 명령어의 실행
  - ✓CCR ← CCRs, PC ← R6s