Multi-cycle CPU 설계 문서

목차

---

1. 개요

1.1 설계 목표

- TSC(Teaching Simple Computer) 마이크로컴퓨터를 위한 16비트 Multi-cycle CPU 구현
- FSM(Finite State Machine) 기반의 5단계 실행 사이클
- 효율적인 명령어 처리를 위한 최적화된 상태 전이

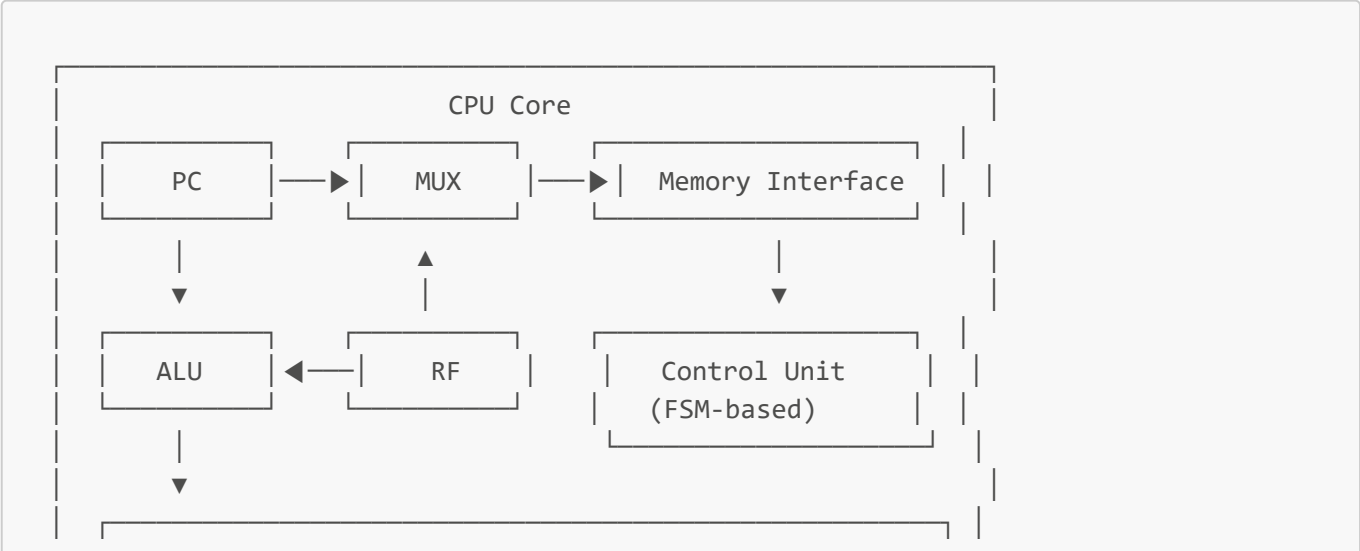
1.2 주요 특징

- **아키텍처**: 16비트 RISC 구조
- **레지스터**: 4개의 범용 레지스터 ($0~$3)
- **명령어 형식**: R-type, I-type, J-type 지원
- **메모리 인터페이스**: 16비트 주소/데이터 버스

---

2. 시스템 구조

2.1 전체 블록 다이어그램



```
  │  │  Internal Registers: IR, MDR, A, B, ALUOut                  │ │
  │  └─────────────────────────────────────────────────────────────┘ │
  └──────────────────────────────────────────────────────────────────┘
```

2.2 인터페이스 신호

신호명	방향	비트 폭	설명
clk	Input	1	시스템 클럭
reset_n	Input	1	Active-low 리셋 신호
readM	Output	1	메모리 읽기 요청
writeM	Output	1	메모리 쓰기 요청
address	Output	16	메모리 주소
data	Inout	16	양방향 데이터 버스
inputReady	Input	1	데이터 준비 완료 신호
num_inst	Output	16	실행된 명령어 수
output_port	Output	16	WWD 출력
is_halted	Output	1	CPU 정지 상태

---

3. 주요 구성 요소

3.1 내부 레지스터

레지스터	비트 폭	용도
PC	16	Program Counter
IR	16	Instruction Register
MDR	16	Memory Data Register
A	16	Register File 출력 임시 저장 (rs)
B	16	Register File 출력 임시 저장 (rt)
ALUOut	16	ALU 연산 결과 임시 저장

3.2 Register File

- **구성**: 4개의 16비트 레지스터 ($0, $1, $2, $3)
- **읽기 포트**: 2개 (비동기)
- **쓰기 포트**: 1개 (동기, rising edge)
- **특수 용도**: $2는 Link 명령어의 return address 저장용

3.3 ALU (Arithmetic Logic Unit)

OP코드	연산	설명
0000	ADD	A + B + Cin
0001	SUB	A - B - Cin
0010	PASS B	B (LHI용)
0110	NOT	~A
0111	AND	A & B
1000	OR	A | B
1101	SHL	A << 1
1011	SHR	A >>> 1 (산술)
1110	TCP	~A + 1 (2의 보수)

---

4. FSM 상태별 동작

4.1 상태 정의

```
STATE_IF  (000): Instruction Fetch
STATE_ID  (001): Instruction Decode
STATE_EX  (010): Execute
STATE_MEM (011): Memory Access
STATE_WB  (100): Write Back
```

4.2 IF (Instruction Fetch)

주요 동작:

1. 메모리에서 명령어 읽기: `Memory[PC] → IR`
2. PC 증가: `PC + 1 → PC`
3. `inputReady` 신호 대기

활성화된 제어 신호:

- `readM = 1`: 메모리 읽기
- `IRWrite = 1`: IR 업데이트
- `ALUSrcA = 0, ALUSrcB = 01`: PC + 1 계산
- `PCWrite = 1`: PC 업데이트

4.3 ID (Instruction Decode)

주요 동작:

1. 명령어 해독 및 레지스터 읽기
2. `RF[rs] → A`, `RF[rt] → B`

3. Branch target 계산: `PC + sign_ext(imm) → ALUOut`

즉시 완료 명령어:

- **Jump**: PC 업데이트 후 IF로
- **WWD**: output_port 설정 후 IF로
- **HLT**: is_halted 설정 후 IF로

4.4 EX (Execute)

명령어별 처리:

명령어 타입	ALU 동작	다음 상태
Branch	조건 검사	IF
R-type ALU	rs op rt	WB
I-type ALU	rs op imm	WB
Load/Store	rs + offset	MEM

4.5 MEM (Memory Access)

동작:

- **Load**: `Memory[ALUOut] → MDR`
- **Store**: `B → Memory[ALUOut]`

4.6 WB (Write Back)

동작:

- 연산 결과를 Register File에 저장
- 목적지 레지스터 선택 (rd 또는 rt)

---

5. 명령어 처리 분석

5.1 명령어 형식

R-type (Register)

```
[15:12] [11:10] [9:8] [7:6] [5:0]
opcode    rs     rt    rd   func
```

I-type (Immediate)

```
[15:12] [11:10] [9:8] [7:0]
opcode    rs     rt    imm
```

J-type (Jump)

```
[15:12] [11:0]
opcode  target
```

5.2 명령어별 실행 사이클

명령어	사이클 수	실행 경로	특이사항
ADD, SUB, AND, OR	4	IF→ID→EX→WB	rd = rs op rt
ADI, ORI	4	IF→ID→EX→WB	rt = rs op imm
LHI	4	IF→ID→EX→WB	rt = {imm, 8'b0}
LWD	5	IF→ID→EX→MEM→WB	rt = Memory[rs+offset]
SWD	4	IF→ID→EX→MEM	Memory[rs+offset] = rt
BNE, BEQ, BGZ, BLZ	3	IF→ID→EX	조건부 분기
JMP	2	IF→ID	PC = {PC[15:12], target}
JAL	2	IF→ID	$2 = PC; JMP
JPR	2	IF→ID	PC = rs
JRL	2	IF→ID	$2 = PC; JPR

5.3 분기 조건

명령어	조건	설명
BNE	rs ≠ rt	Branch Not Equal
BEQ	rs = rt	Branch Equal
BGZ	rs > 0	Branch Greater than Zero
BLZ	rs < 0	Branch Less than Zero

---

6. 데이터 경로

6.1 PC 업데이트 경로

```
PCSource 선택:
┌──────────┬──────────────────────┐
│ PCSource │ PC 다음 값           │
├──────────┼──────────────────────┤
│    00    │ alu_result (PC+1)    │
```

```
|    01    | ALUOut (branch)       |
|    10    | {PC[15:12], target}   |
|    11    | read_data1 (rs)       |
└──────────┴───────────────────────┘
```

6.2 ALU 입력 선택

ALU Input A:

- `ALUSrcA = 0`: PC
- `ALUSrcA = 1`: A register (rs)

ALU Input B:

```
┌──────────┬──────────────────────┐
| ALUSrcB  | 선택되는 값          |
├──────────┼──────────────────────┤
|    00    | B register (rt)      |
|    01    | 상수 1               |
|    10    | Extended immediate   |
|    11    | {imm, 8'b0}          |
└──────────┴──────────────────────┘
```

6.3 Register File 쓰기 경로

쓰기 주소 선택:

- Link 명령어 (JAL, JRL): 항상 $2
- R-type: rd 필드
- I-type: rt 필드

쓰기 데이터 선택:

- `MemtoReg = 1`: MDR (Load 명령어)
- Link 명령어: PC
- 기타: ALUOut

---

7. 제어 신호

7.1 주요 제어 신호

신호명	기능
PCWrite	PC 무조건 업데이트
PCWriteCond	조건부 PC 업데이트
IorD	메모리 주소 선택 (0: PC, 1: ALUOut)

신호명	기능
MemRead	메모리 읽기 활성화
MemWrite	메모리 쓰기 활성화
IRWrite	IR 업데이트 활성화
MemtoReg	RF 쓰기 데이터 선택
RegDst	RF 쓰기 주소 선택
RegWrite	RF 쓰기 활성화

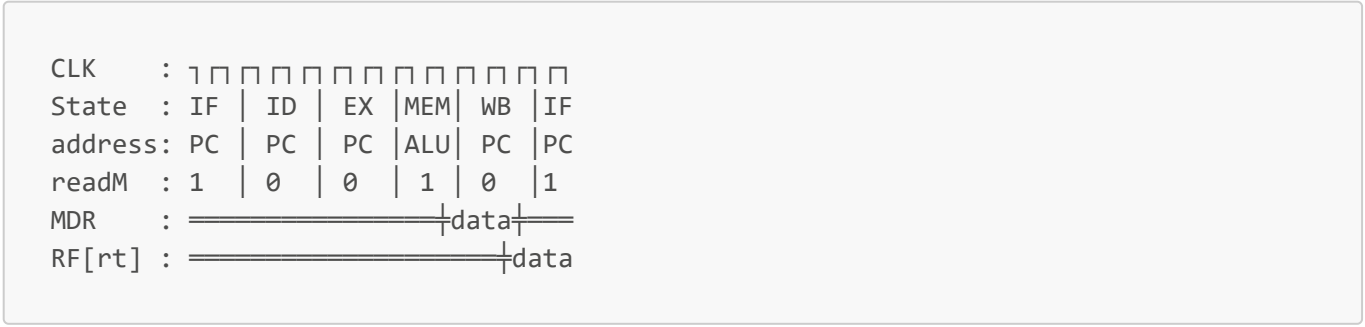
7.2 상태별 제어 신호

상태	주요 활성 신호	설명
IF	MemRead, IRWrite, PCWrite	명령어 인출
ID	명령어별 상이	해독 및 분기
EX	ALUSrcA=1, 명령어별 ALUOp	ALU 연산
MEM	IorD=1, MemRead/Write	메모리 접근
WB	RegWrite, MemtoReg	결과 저장

8. 타이밍 다이어그램

8.1 기본 명령어 실행 (ADD)

8.2 Load 명령어 (LWD)



9. 구현 세부사항

9.1 특수 처리 사항

Immediate 확장

- **ORI**: Zero extension 사용
- **기타 I-type**: Sign extension 사용

Link 명령어 처리

- Return address는 항상 $2에 저장
- ID 단계에서 PC 값 저장 (이미 +1된 상태)

명령어 카운터

- 각 명령어의 마지막 실행 단계에서 증가
- 명령어 타입별로 다른 시점에 카운트

9.2 최적화 특징

1. **조기 분기 결정**: Branch target을 ID 단계에서 미리 계산
2. **최소 사이클**: Jump류 명령어는 2사이클에 완료
3. **효율적인 상태 전이**: 불필요한 상태 방문 최소화

9.3 메모리 인터페이스

- **양방향 데이터 버스**:
  - 쓰기: `data = B`
  - 읽기: `data = Z` (High impedance)
- **inputReady 동기화**: 메모리 읽기 시 대기

부록: 명령어 세트 요약

A.1 R-type 명령어

명령어	Opcode	Function	동작
ADD	0xF	0x00	rd = rs + rt
SUB	0xF	0x01	rd = rs - rt
AND	0xF	0x02	rd = rs & rt
OR	0xF	0x03	rd = rs | rt
NOT	0xF	0x04	rd = ~rs
TCP	0xF	0x05	rd = ~rs + 1
SHL	0xF	0x06	rd = rs << 1

명령어	Opcode	Function	동작
SHR	0xF	0x07	rd = rs >> 1

A.2 I-type 명령어

명령어	Opcode	동작
ADI	0x4	rt = rs + sign_ext(imm)
ORI	0x5	rt = rs | zero_ext(imm)
LHI	0x6	rt = {imm, 8'b0}
LWD	0x7	rt = Memory[rs + offset]
SWD	0x8	Memory[rs + offset] = rt

A.3 분기 및 점프 명령어

명령어	Opcode	동작
BNE	0x0	if (rs ≠ rt) PC = PC + offset
BEQ	0x1	if (rs = rt) PC = PC + offset
BGZ	0x2	if (rs > 0) PC = PC + offset
BLZ	0x3	if (rs < 0) PC = PC + offset
JMP	0x9	PC = {PC[15:12], target}
JAL	0xA	$2 = PC; JMP

본 문서는 TSC Multi-cycle CPU의 설계 및 구현을 위한 기술 문서입니다.