Introduction to Programmable Processor

CPU

Yoonjin Kim

Full Professor

Dept. of Computer Science
Sookmyung Women's University

Outline

processer의 중요한 부분 꼭 알기!

- Digital circuit design hierarchy ⇒ 디지털 회로 디자인 계층
- Overview of programmable processor
- Basic architecture of programmable processor
- RTL-design method for programmable processor (Register Transfer Level) ⇒ programmable processor 모델을 단순화하여 RTL로 디자인하면됨.
- Four-instruction processor example

4개 명령어로 진행한 프로세서 예시

(micro processer로 구현한 예시 vs 동일한 알고리즘을 하드웨어로 만든것) 차이점! 까지

+ 어셈블리어로 구현한 것
+ 하드웨어로 한 것 비교까지

Definition of Terms (용어 정의)

"동의어"

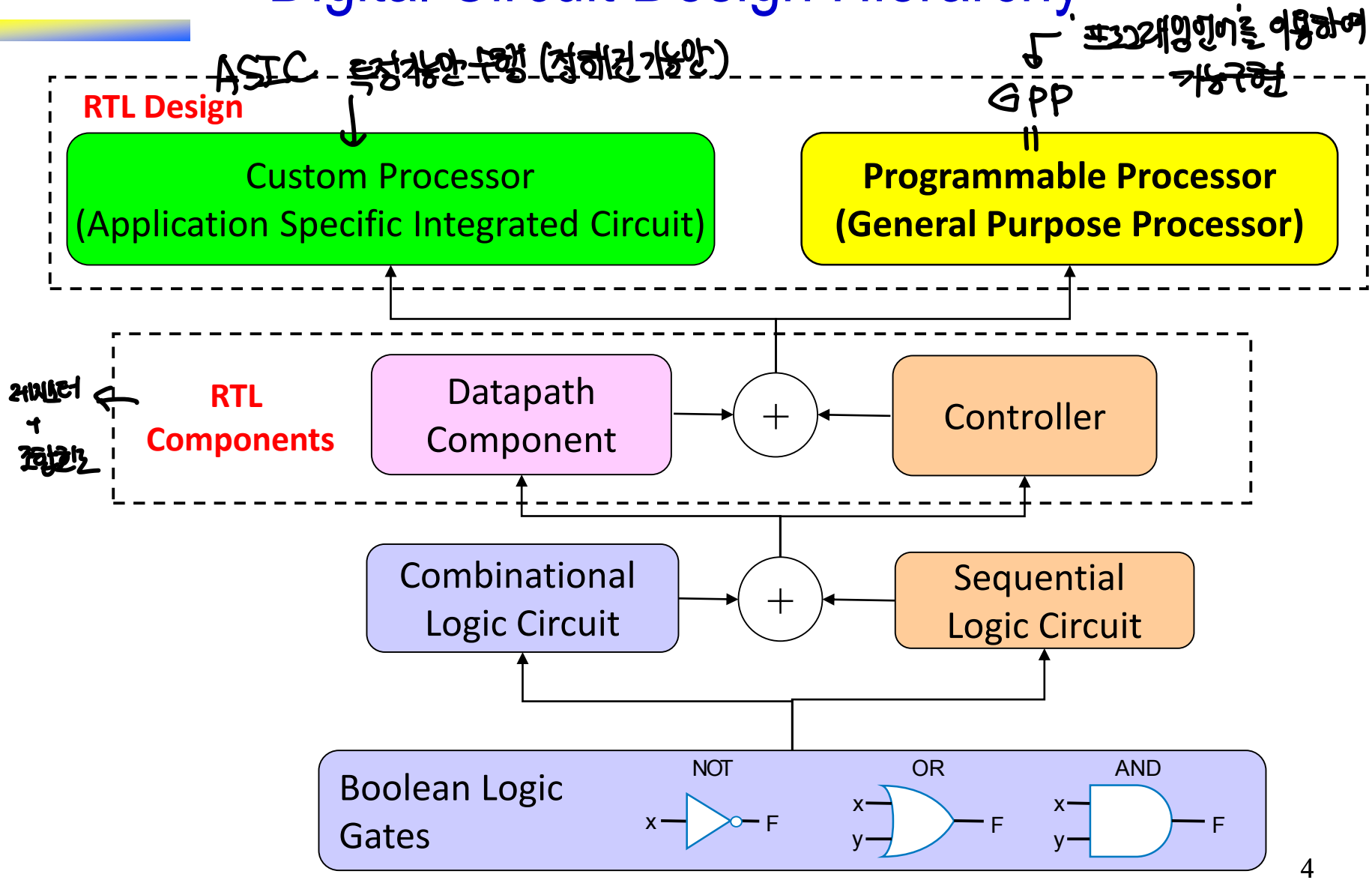
- Synonym
  - CPU (Central Processing Unit)
  - GPP (General Purpose Processor)
  - Programmable processor
  - Microprocessor

다같은 말임.

- Synonym
  - Customized digital circuit
  - Custom processor
  - ASIC (Application Specific Integrated Circuit)
  - Dedicated hardware
  - Hardware accelerator

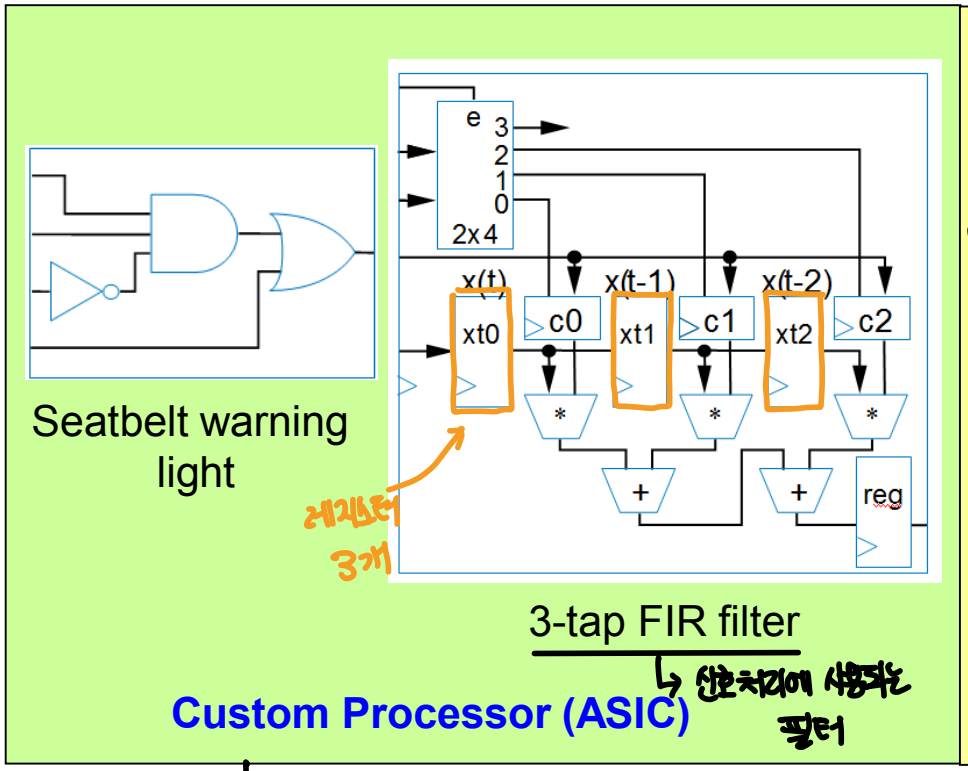
다같은 말임.

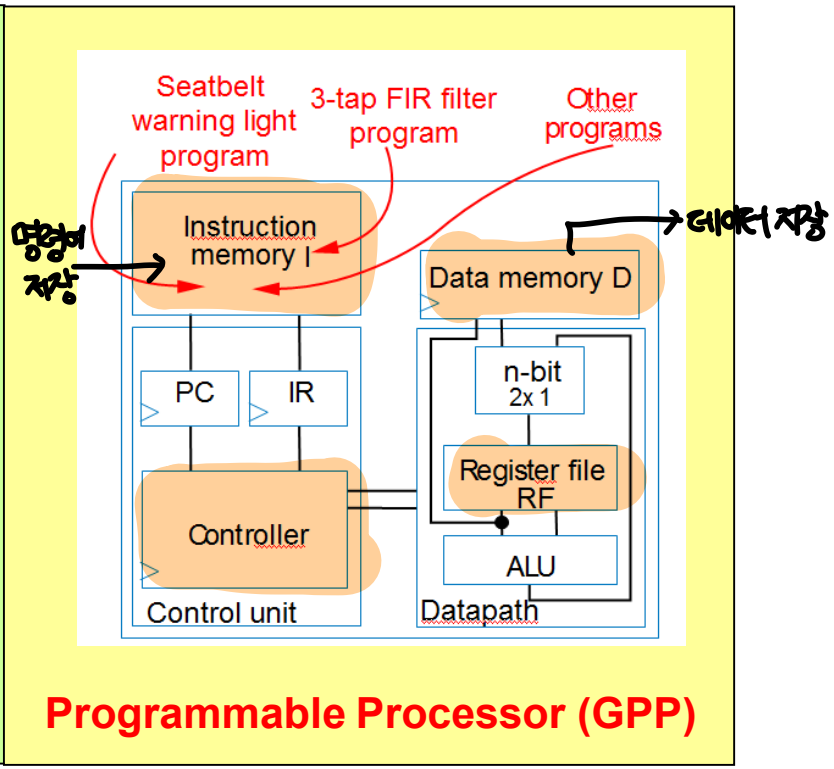
Digital Circuit Design Hierarchy



Overview of Programmable Processor

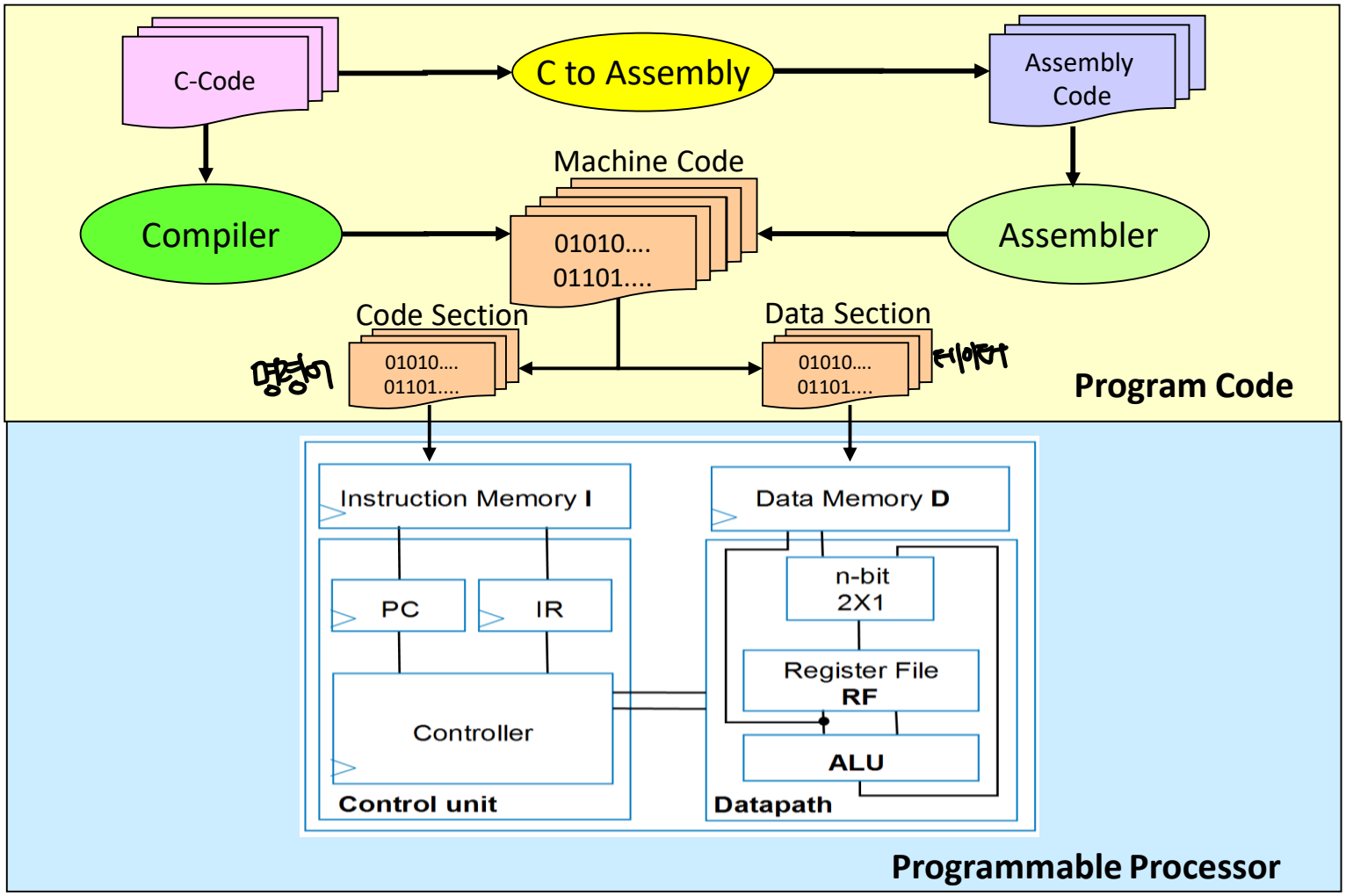
- Programmable Processor (General Purpose Processor)
  - Different processing tasks can be programmed and run on GPP.





Overview of Programmable Processor

- From Program Code to Programmable Processor



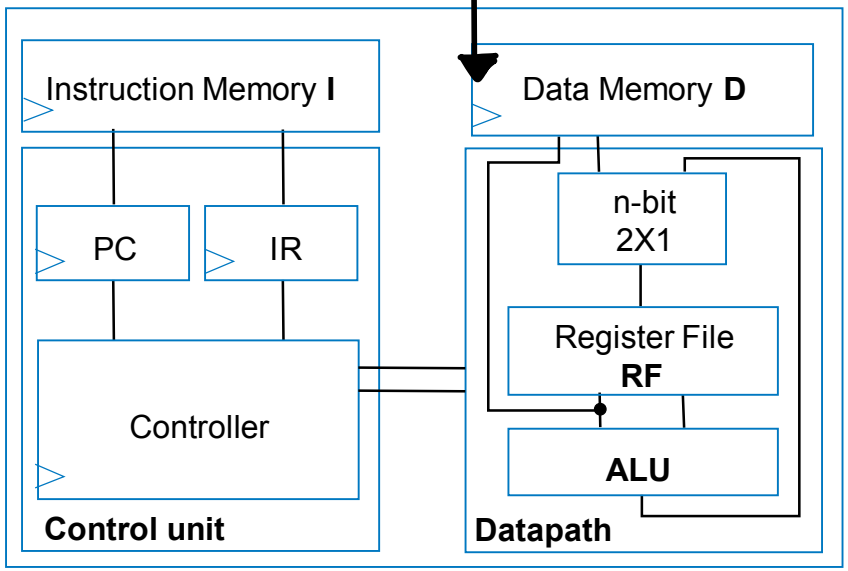
Overview of Programmable Processor

- Well-Known Common Programmable Processors
  - Desktop CPU: Intel-Pentium, AMD-Athlon, Sun Microsys-SPARC
  - Embedded Processor: ARM-Cortex, Intel-Atom, IBM-PowerPC.

⇒ 데스크탑, 휴대폰 등에 들어감

- This Chapter Introduces
  - A very simple programmable processor as shown in the below figure.
  - It's very Instructive understanding the principle of GPP.
  - Real processors can be much more complex.

⇒ 실제 프로세서는 더욱 복잡.

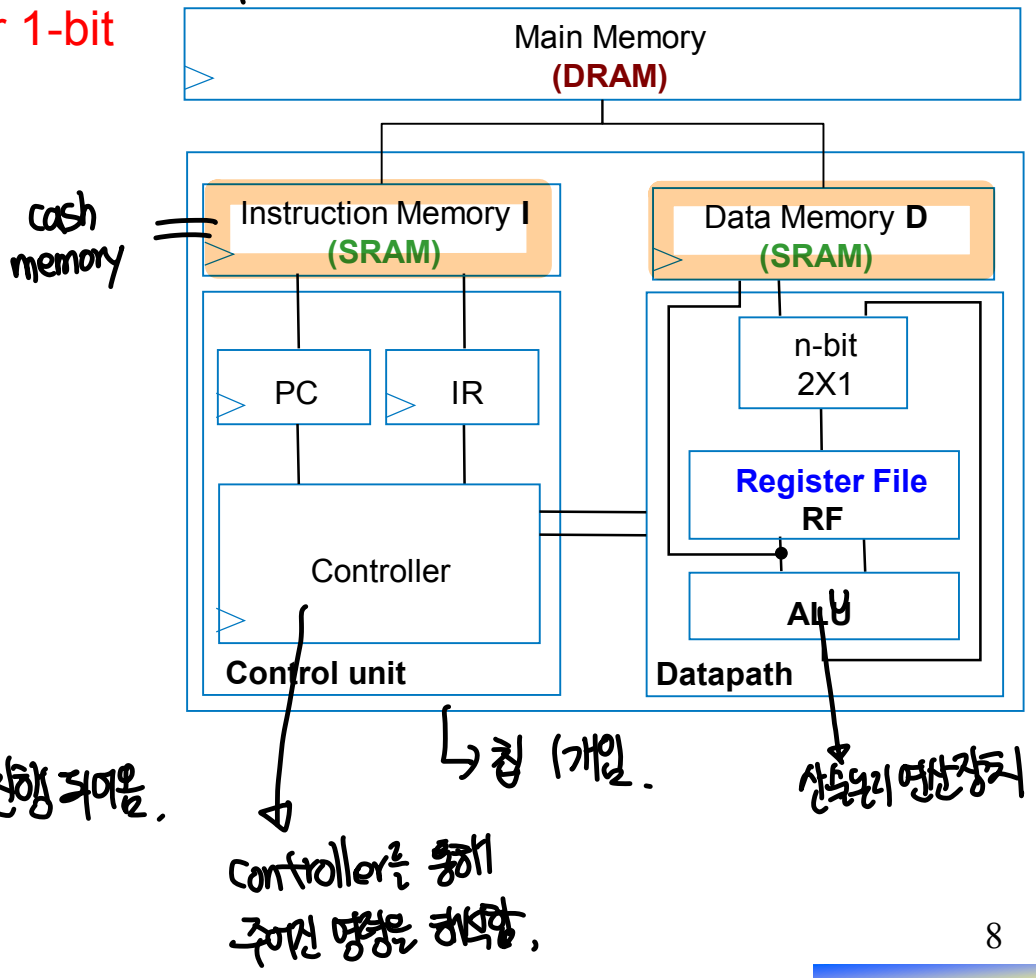


Basic Architecture of Programmable Processor

- Memory Hierarchy Aspect

GPP

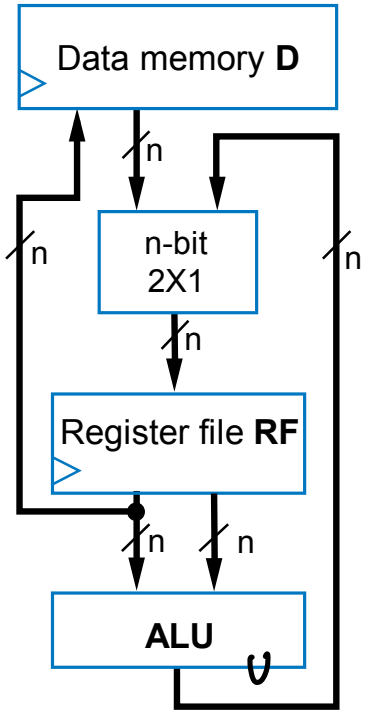
- **Register File** – 38 transistors for 1-bit
  - Fastest
  - But biggest size
  - Very expensive
- **SRAM** – 6 transistors for 1-bit
  - Fast
  - More compact than register file
  - Expensive
- **DRAM** – 1 transistor for 1-bit
  - Slowest
    - And refreshing takes time
  - But very compact
  - Cheap



Basic Architecture of Programmable Processor

2 - Datapath Aspect

- Processing on GPP consists of:
  - Loading some data 데이터 읽어오기
  - Transforming that data 데이터 가져오기
  - Storing that data 데이터 저장(쓰기)
- *Datapath*: Useful circuit in a programmable processor
  - Can read/write data memory, where main data exists.
  - Has register file to hold data locally.
  - Has ALU to transform local data.



Datapath

ALU에는 메모리가 레지스터만 연결되어있다.
(산술논리연산장치)

Basic Architecture of Programmable Processor
- Basic Datapath Operations

(Data path 중심)

- ① **Load operation:** Load data from data memory to RF
- ② **ALU operation:** Transforms data by passing one or two RF register values through ALU, performing operation (ADD, SUB, AND, OR, etc.), and writing back into RF.
- ③ **Store operation:** Stores RF register value back into data memory
- Each operation can be done in one clock cycle

↳ RF에서 ALU로 가져오는 것을 ALU에서 쓰는 것.

↳ 결과물을 다시 메모리에 가져다 쓰는 것.

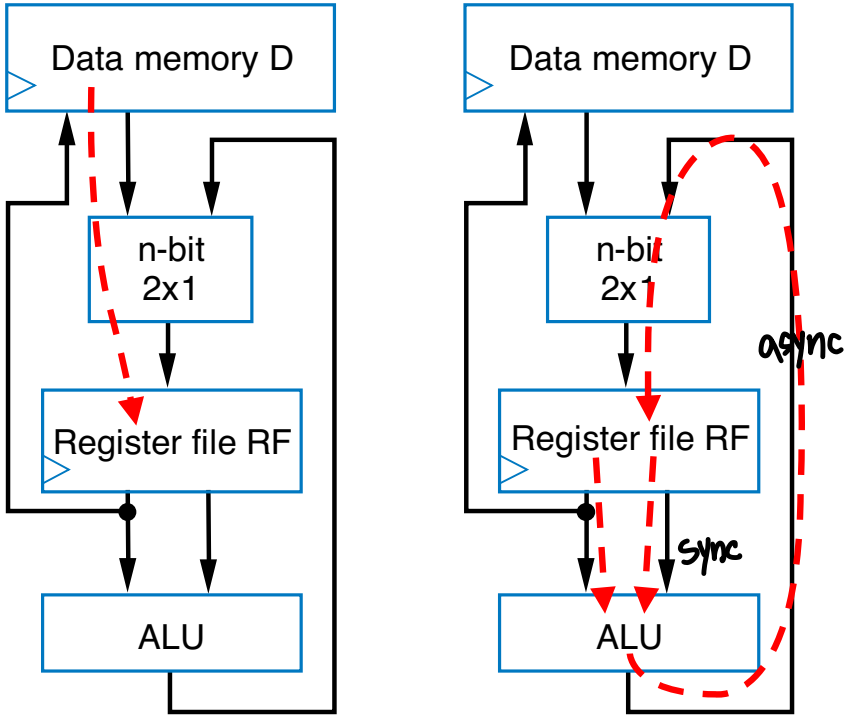
이 각각의 동작은 1 CLK에서 일어남.

⇓

다음 3가지 동작이 제일 기본이 되는 내용.

⇓

3가지 동작이 더 세분화되어 확장됨.



Load operation ALU operation Store operation

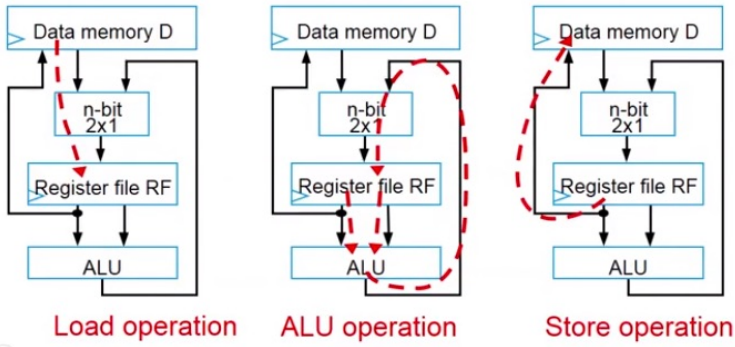
모든 SW 코드는

(Load / ALU / Store) 이 세개로 쪼개져서 CPU가 수행하는 것임.

Basic Datapath Operations

↳ data path가 1 cycle 안에 가능한가?

- Q: Which are valid *single-cycle operations* for given datapath?
  - Move D[1] to RF[1] (i.e., RF[1] = D[1])
    - A: YES – That's a load operation
  - Store RF[1] to D[9] and store RF[2] to D[10] ⇒ 두개의 store … 2 cycle이상 걸림.
    - A: NO – Requires two separate store operations
  - Add D[0] plus D[1], store result in D[9] ⇒ load + ALU + store까지하면 많은 cycle이 필요함.
    - A: NO – ALU operation (ADD) only works with RF. Requires two load operations (e.g., RF[0]=D[0]; RF[1]=D[1], an ALU operation (e.g., RF[2]=RF[0]+RF[1]), and a store operation (e.g., D[9]=RF[2])



10

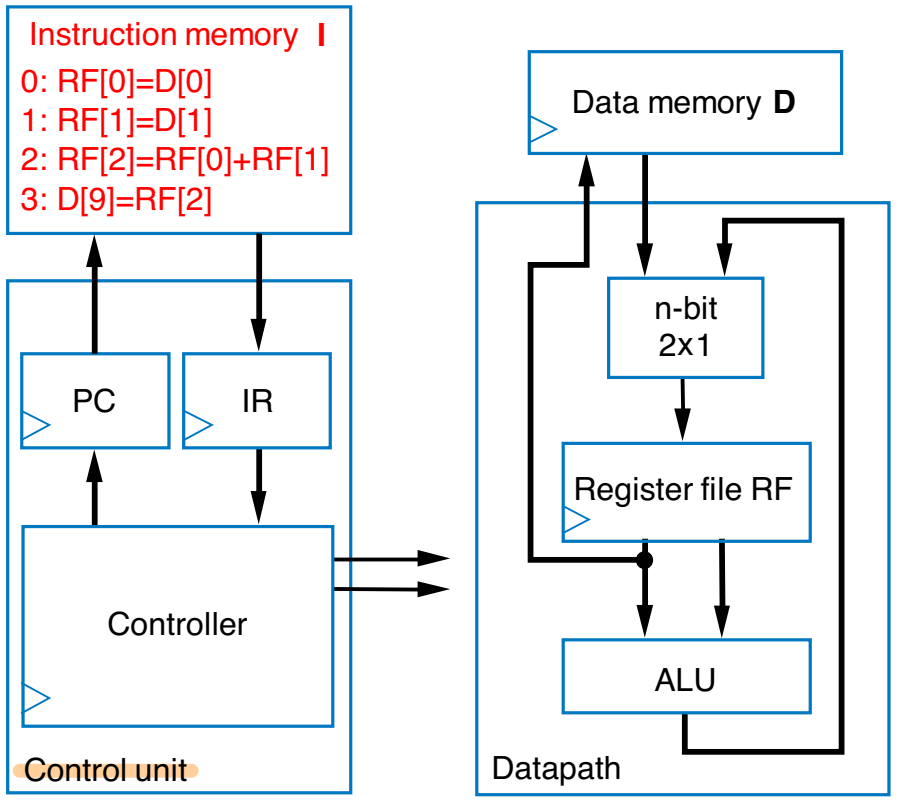
Store는 값을 보면 Memory 상 1 clk이 필요.

Q. 왜 3가지 기능으로 나누어서 CPU는 계산을 하는 것인가?

+ 추가 설명 페이지.

Basic Architecture of Programmable Processor - Control Unit

- D[9] = D[0] + D[1] – requires a sequence of four datapath operations:
  - 0: RF[0] = D[0] // Load Operation
  - 1: RF[1] = D[1] // Load Operation
  - 2: RF[2] = RF[0] + RF[1] // ALU Operation
  - 3: D[9] = RF[2] // Store Operation
- Each operation is an *instruction* (명령어)
  - Sequence of instructions – *program*
  - GPP decomposes desired computations into *processor-supported* operations (Load, ALU, Store operation).
  - Store program in *Instruction memory.*
  - *Control unit* reads each instruction and executes it on the datapath
    - PC: Program counter – address of current instruction
    - IR: Instruction register – current instruction



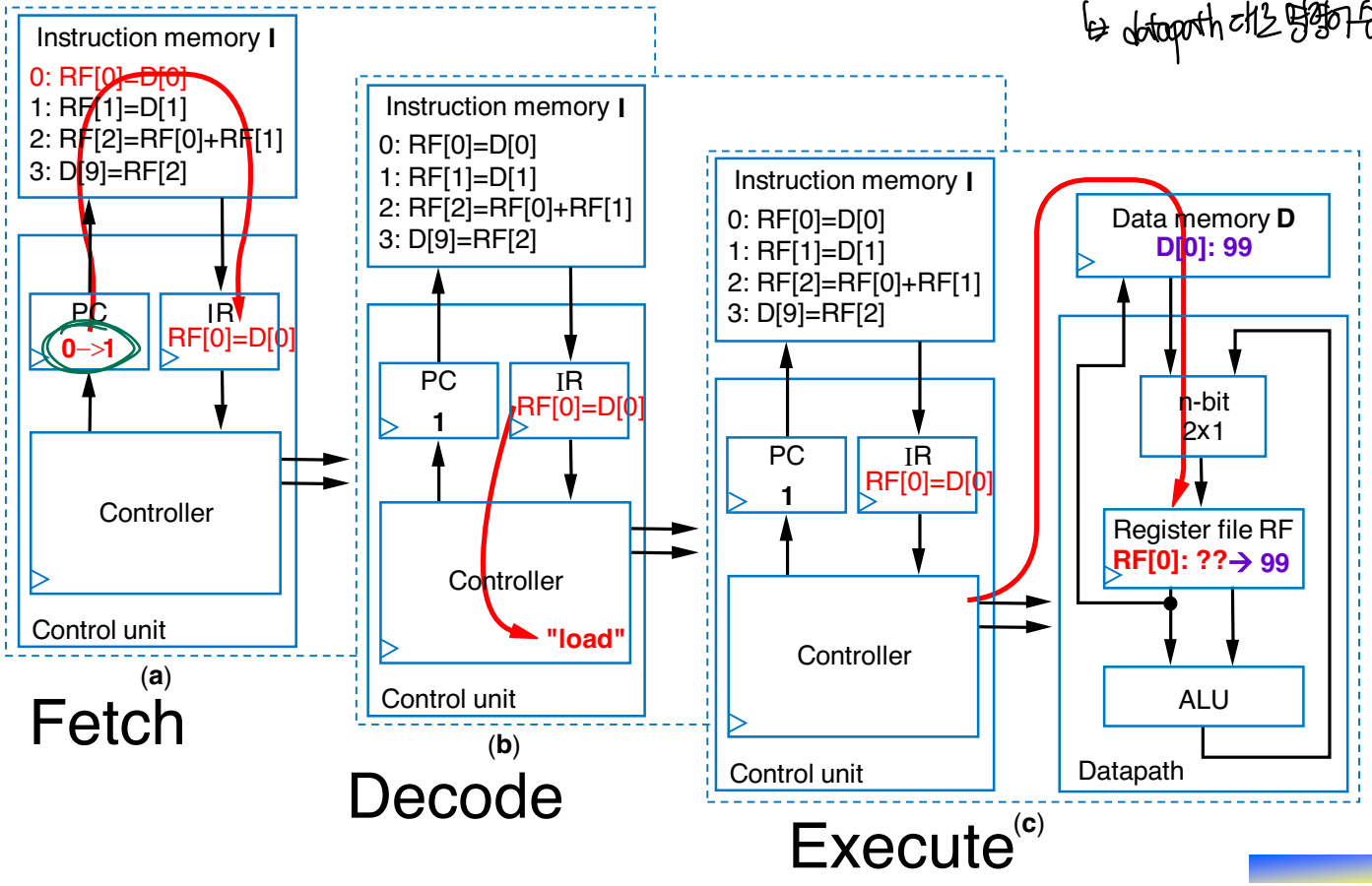
PC와 IR를 대입을 할 수 없음.

프로그램이 현재 어디까지 왔는가

현재 어떤 명령어를 수행하고 있는가

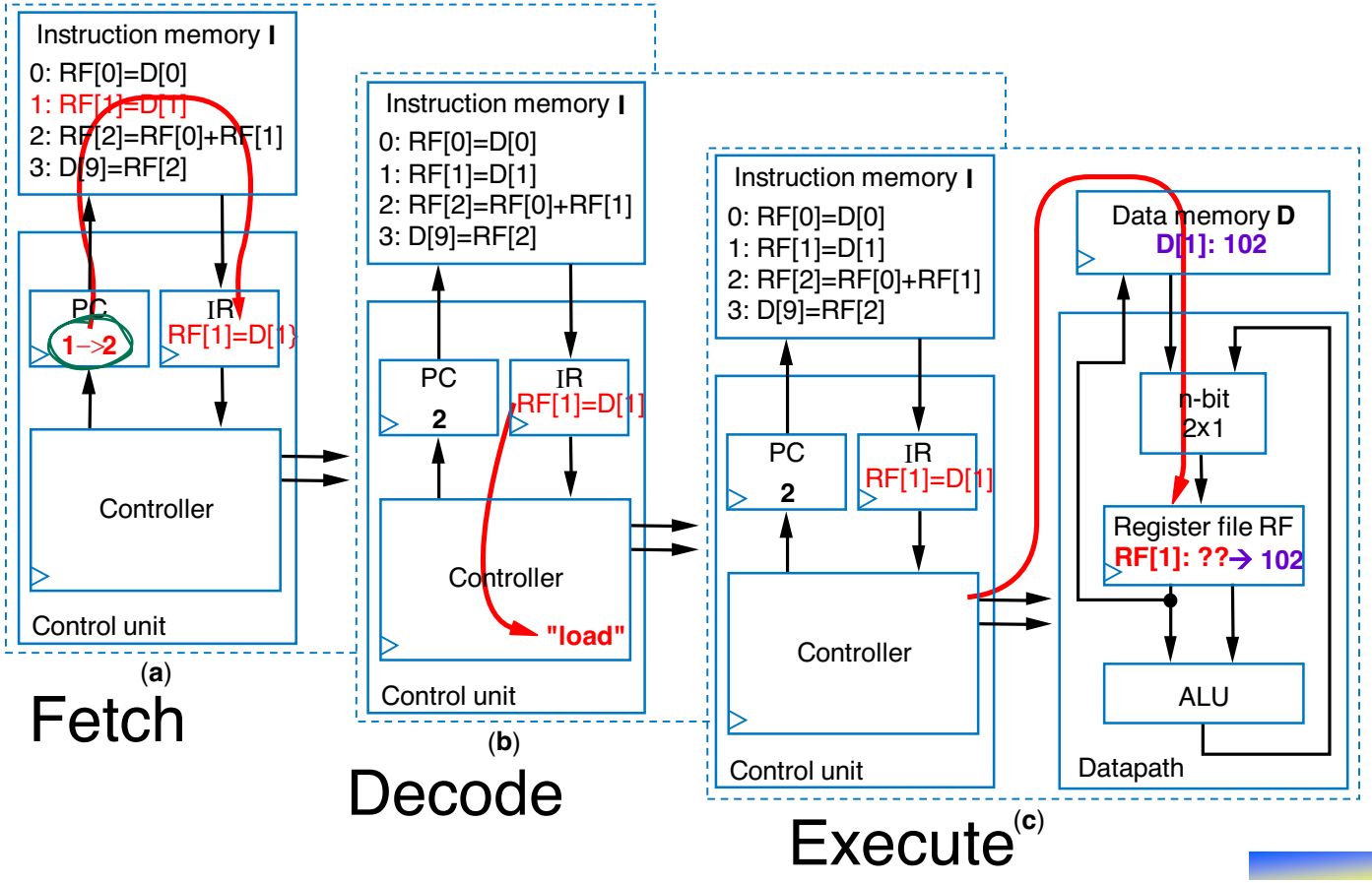
Basic Architecture of Programmable Processor - Control Unit

- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from instruction memory **I**.
  - Decode – Determine the operation and operands of the instruction.
  - Execute – Carry out the instruction's operation using the datapath.



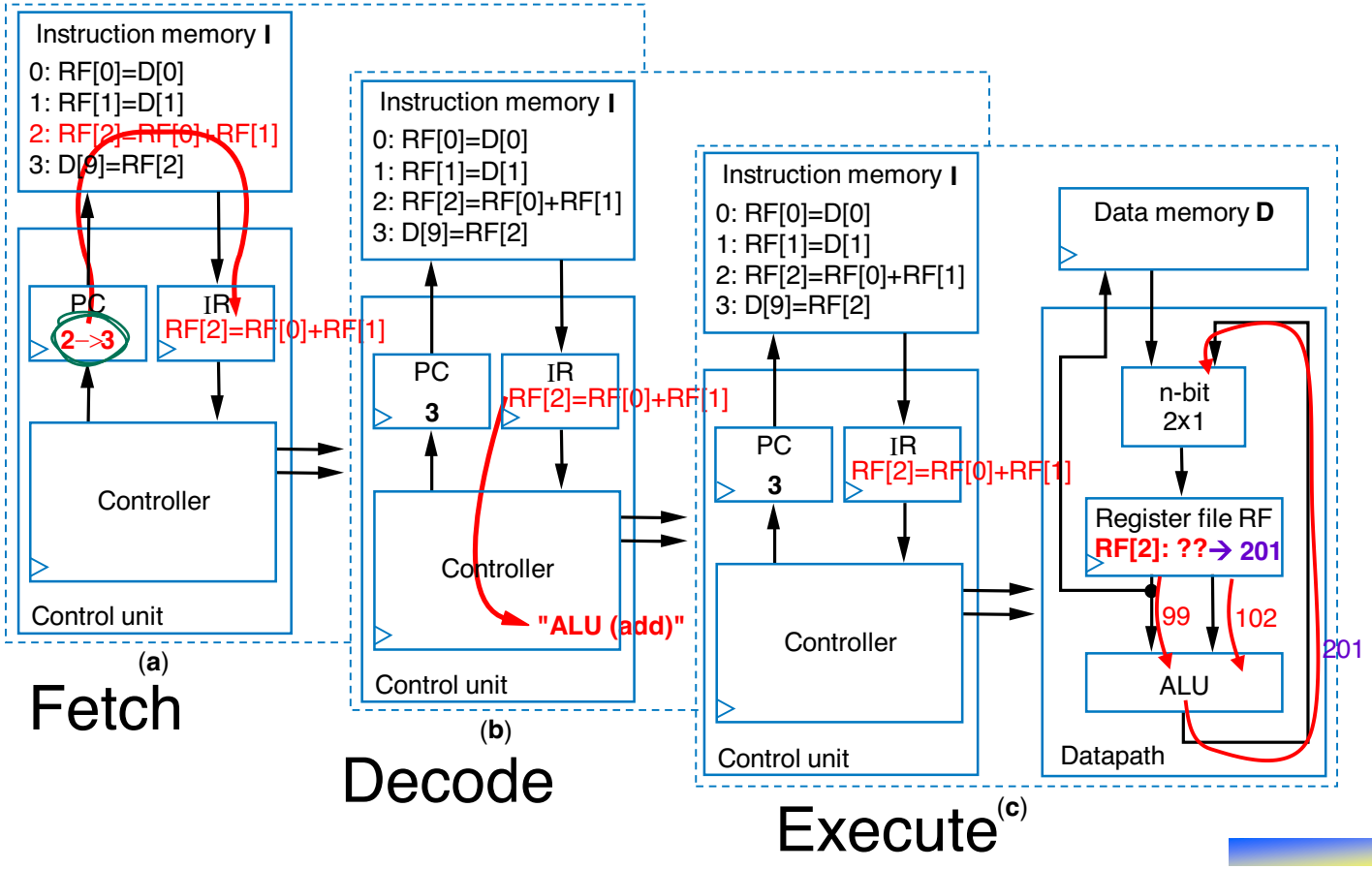
Basic Architecture of Programmable Processor - Control Unit

- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from instruction memory **I**.
  - Decode – Determine the operation and operands of the instruction.
  - Execute – Carry out the instruction's operation using the datapath.



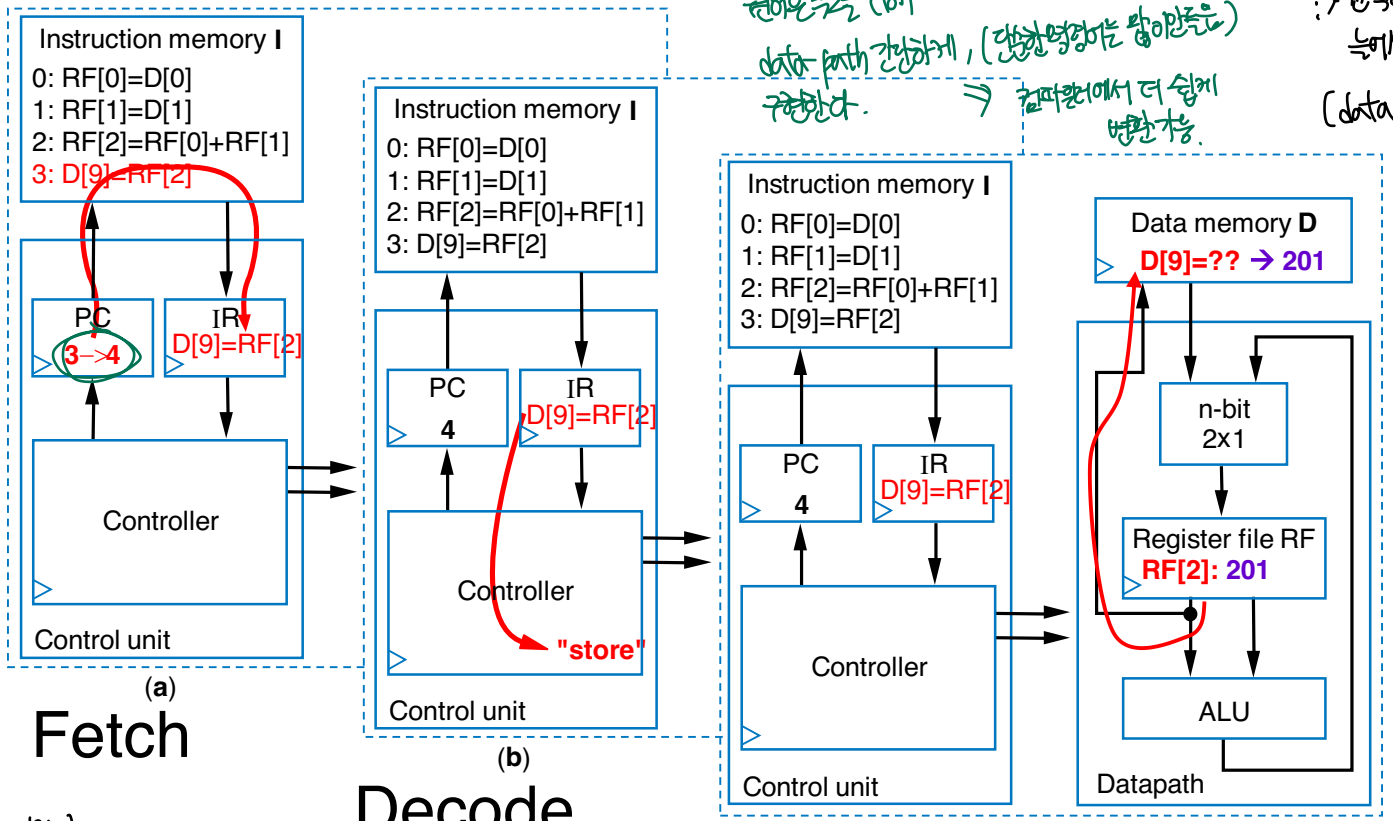
Basic Architecture of Programmable Processor - Control Unit

- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from instruction memory **I.**
  - Decode – Determine the operation and operands of the instruction.
  - Execute – Carry out the instruction's operation using the datapath.



Basic Architecture of Programmable Processor - Control Unit

- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from instruction memory **I**.
  - Decode – Determine the operation and operands of the instruction.
  - Execute – Carry out the instruction's operation using the datapath.



대략적으로 이게 무엇인지 설명할 것임.

RTL Design Method for Programmable Processor

- It's actually same method as custom processor (ASIC) is designed by – process composed of 4 steps.
- However, a preliminary step is required – Define Instruction Set Architecture.
- Instruction Set Architecture [ISA] – List of allowable instructions and their bit-level representation.

명령어 체계

fetch → decode

Step 0 *Define Instruction Set Architecture* [ISA]

	Step	Description
Step 1	*Capture a high-level state machine*	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on bit inputs and outputs.
Step 2	*Create a datapath*	Create a datapath to carry out the data operations of the high-level state machine.
Step 3	*Connect the datapath to a controller*	Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.
Step 4	*Derive the controller's FSM*	Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath.

Four-Instruction Processor Example:
Step 0 – Define Instruction Set Architecture

- ISA for the example processor
  - Instruction length : 16-bit
  - Addressibility of Register File : 16 (bit-width of address port is 4-bit.)
  - Addressibility of Data Memory: 256 (bit-width of address port is 8-bit.)
  - Allowable instructions: Four Instructions (Load, Store, Add, Mult)

ISA for Load/Store Instruction

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode				Register Address				Memory Address							

ISA for Add/Mult Instruction

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode				Register#1 Address				Register#2 Address				Register#3 Address			

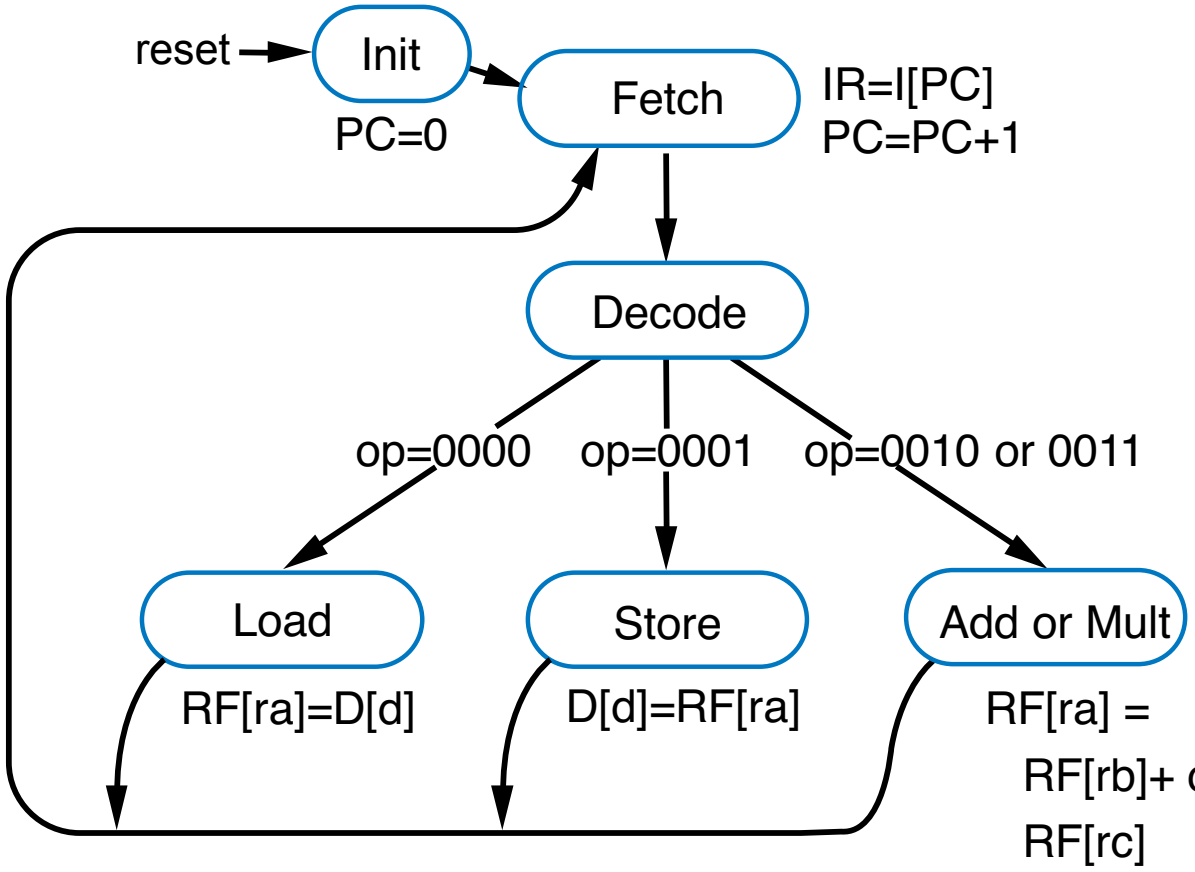
Definition of opcode

opcode	Instruction
0000	Load
0001	Store
0010	Add
0011	Mult

Four-Instruction Processor Example:

Step 1 – Capture a High-Level State Machine

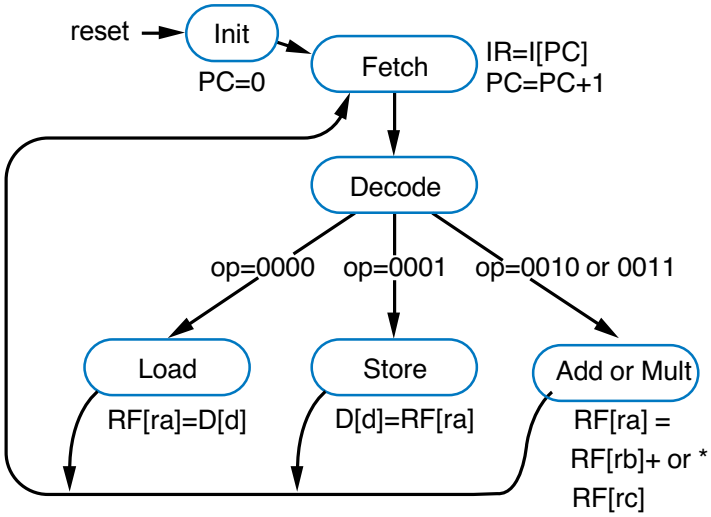
[state Machine]

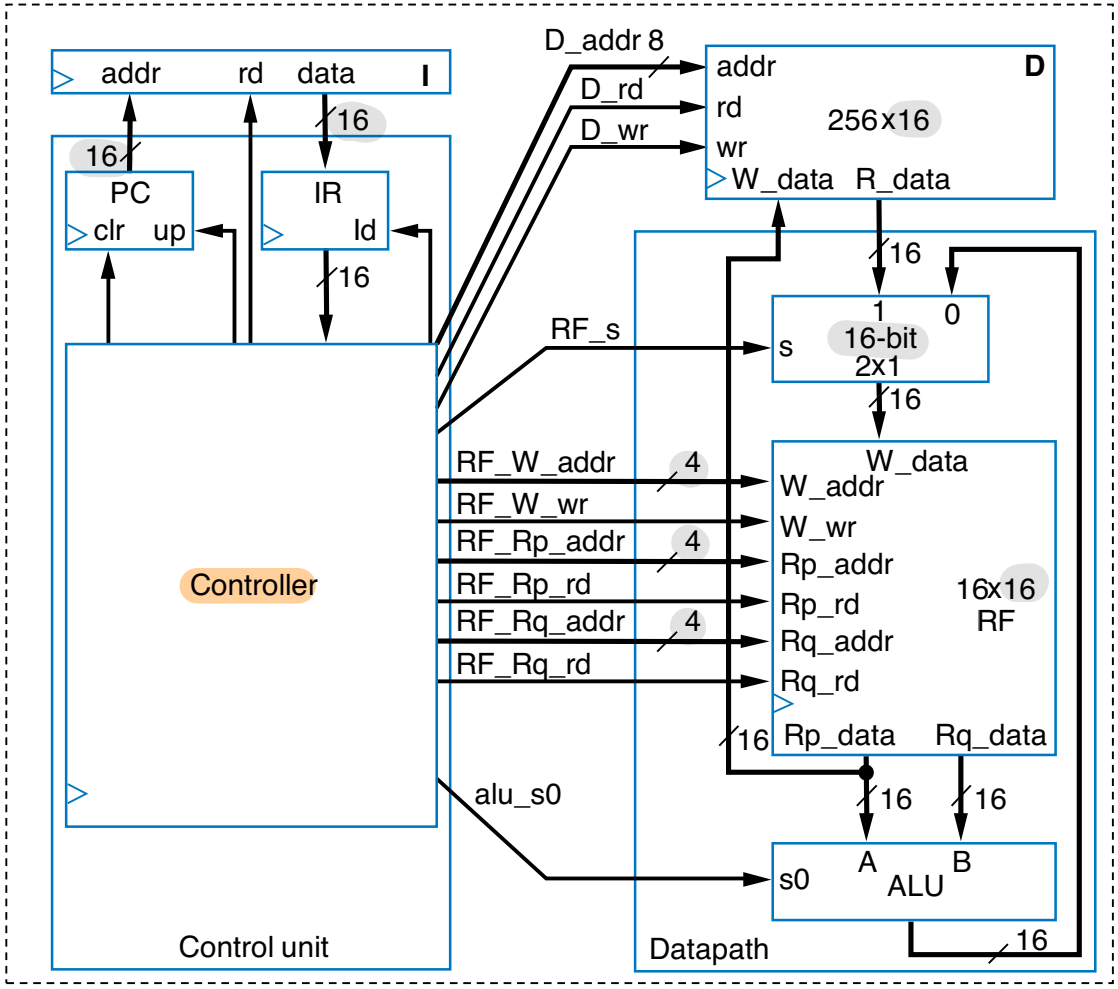


⇒ excute

Four-Instruction Processor Example:

Step 2 and 3 – Create a Datapath and Connect Datapath to a Controller

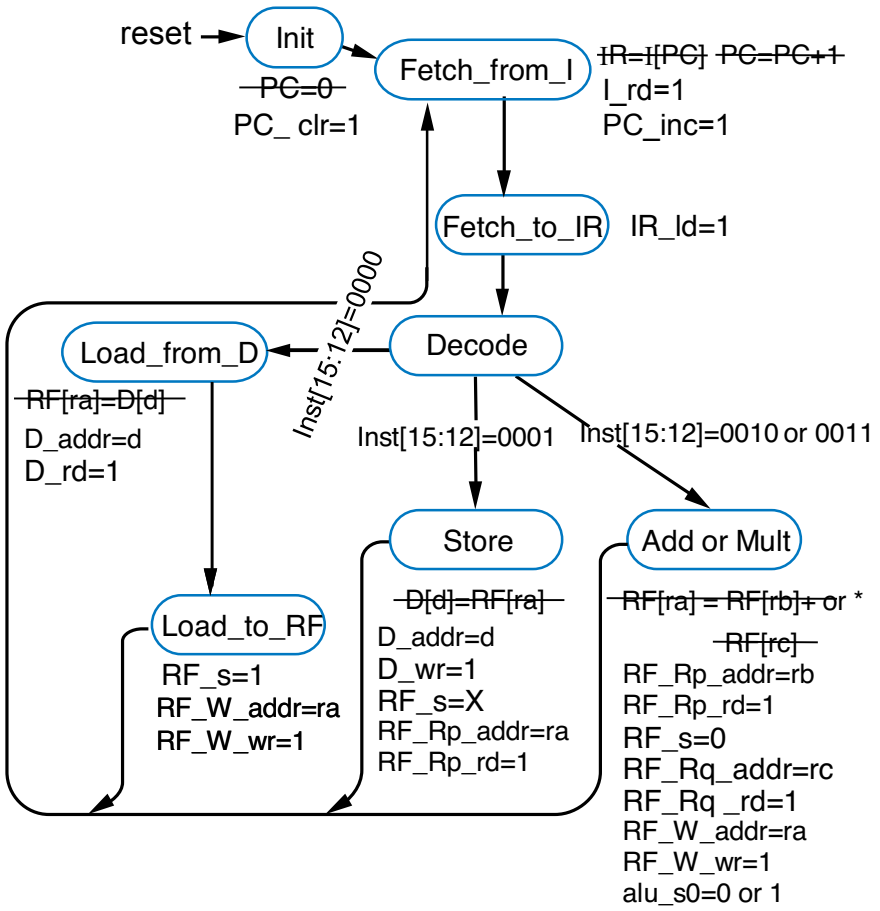


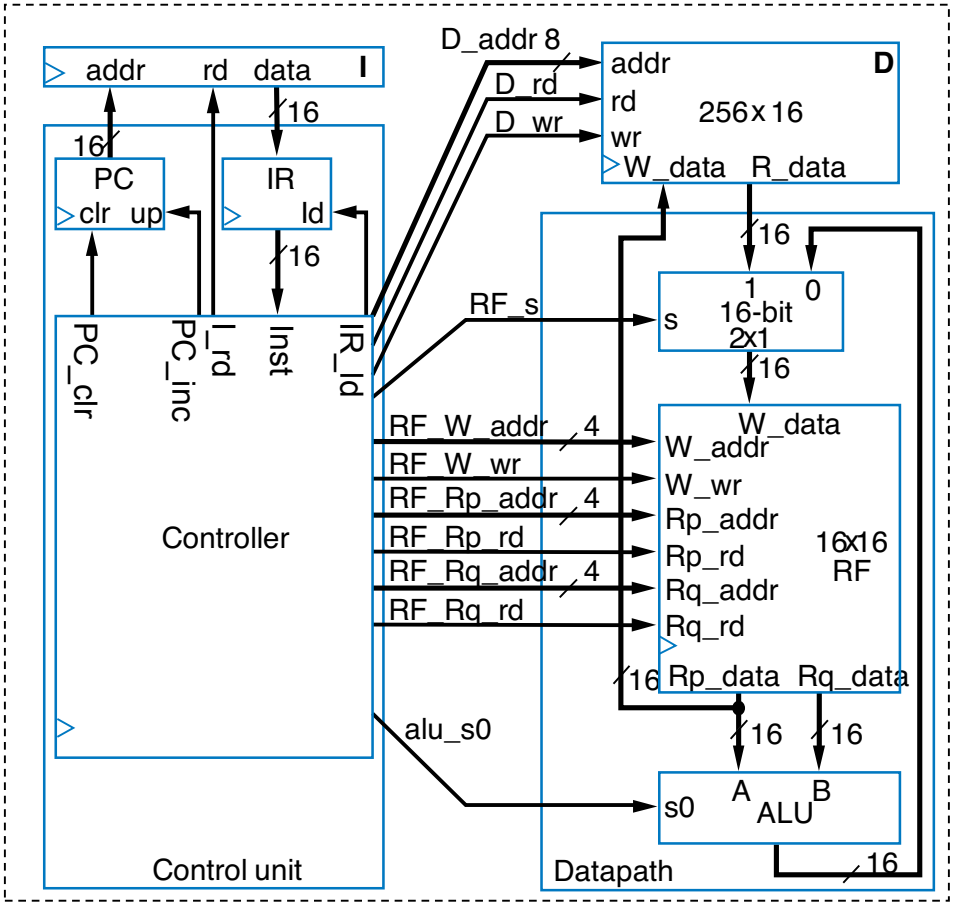


Four-Instruction Processor Example:

Step 4 – Derive the Controller's FSM

[FSM]





Four-Instruction Processor Example:
Critical Path of Four-Instruction Processor

(동작주파수)

- Let's assume that
  - Delay of 16-bit Adder : 2 ns
  - Delay of 16-bit Multiplier: 8 ns
  - Delay of 16-bit 2x1 Multiplexer: 1 ns
  - Wire delay and register setup time/internal delay are ignored.

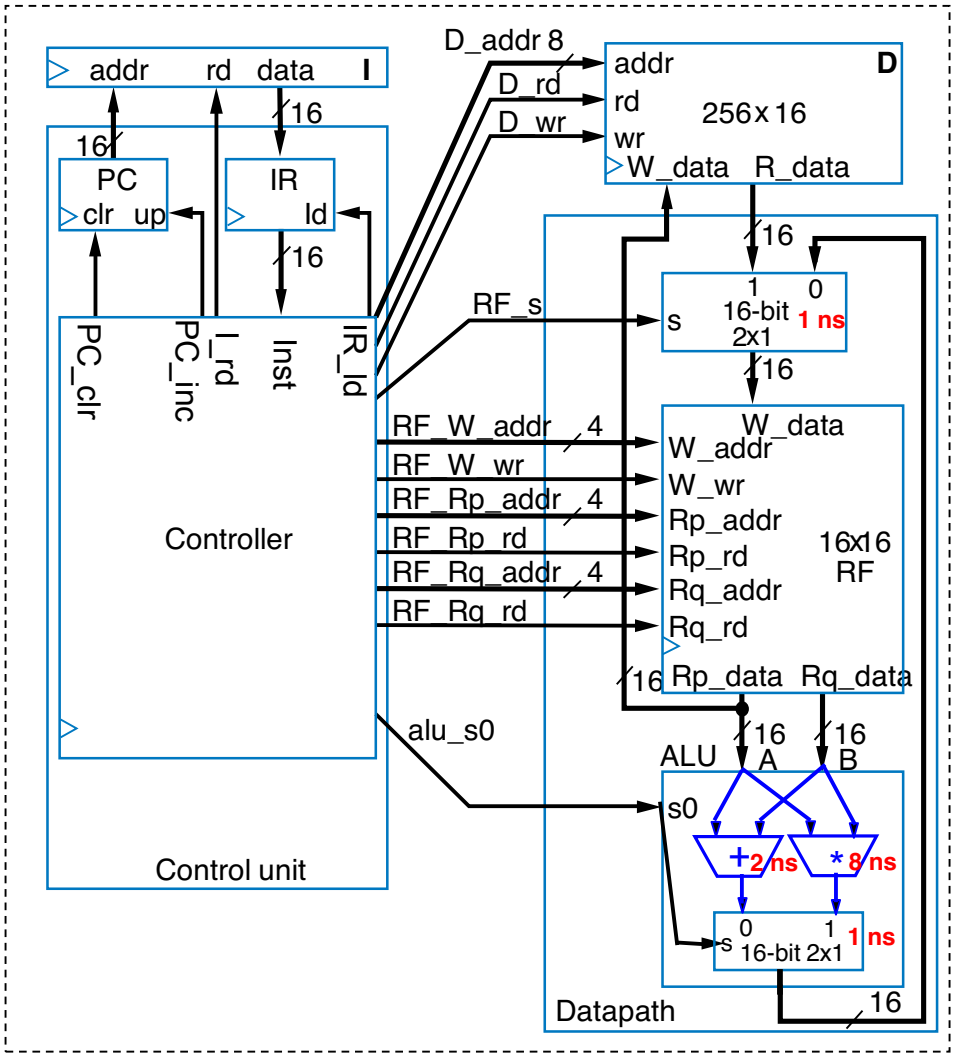
- Critical path ⇒ 가장 오래걸리는 최대 기능 구현시간
  - Multiplier (8 ns) + 2x1 MUX (1ns) + 2x1 MUX (1ns) = 10 ns $= 10 \times 10^{-9}$ s $= 10^{-8}$ s

- Operating Frequency: 100 MHz

동작 주파수 $= \frac{1}{\text{Critical path}} = \frac{1}{10^{-8}\text{s}} = 10^{8}$ Hz $= 100$ MHz

↳ 이것보다 커지면 동작 못함.

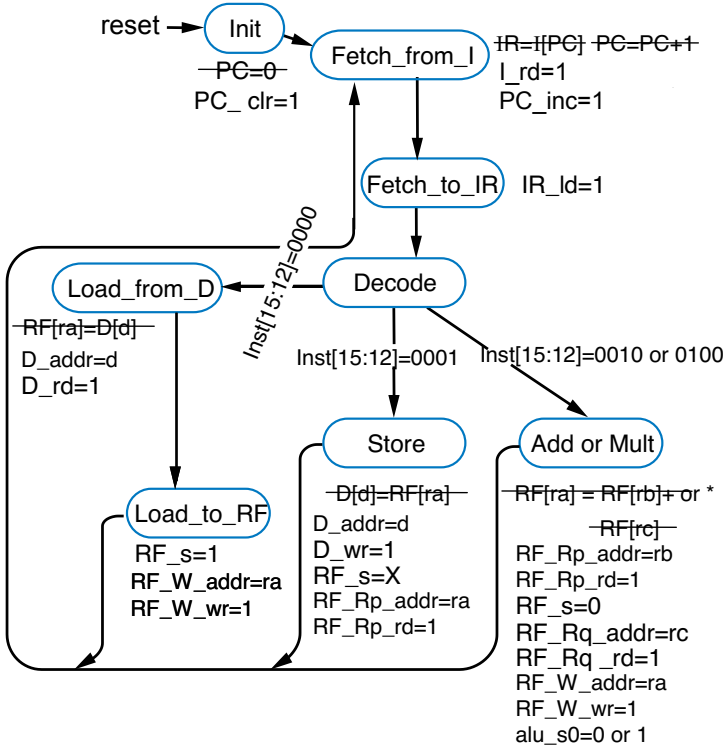
↳ 이것보다 더 작아지게해서 안정적으로 돌아가도록 하겠다.



Four-Instruction Processor Example:
Cycles Per Instruction [CPI]

cycle ⇒ clk이 몇번 변화하는가

- Q: How many cycles are needed to execute instructions using the four-instruction processor – Cycles Per Instruction [CPI]?
- A: Load instruction requires 5 cycles and other instructions requires 4 cycles. why?
  - CPI can be calculated by counting the number of states corresponding to the instruction.

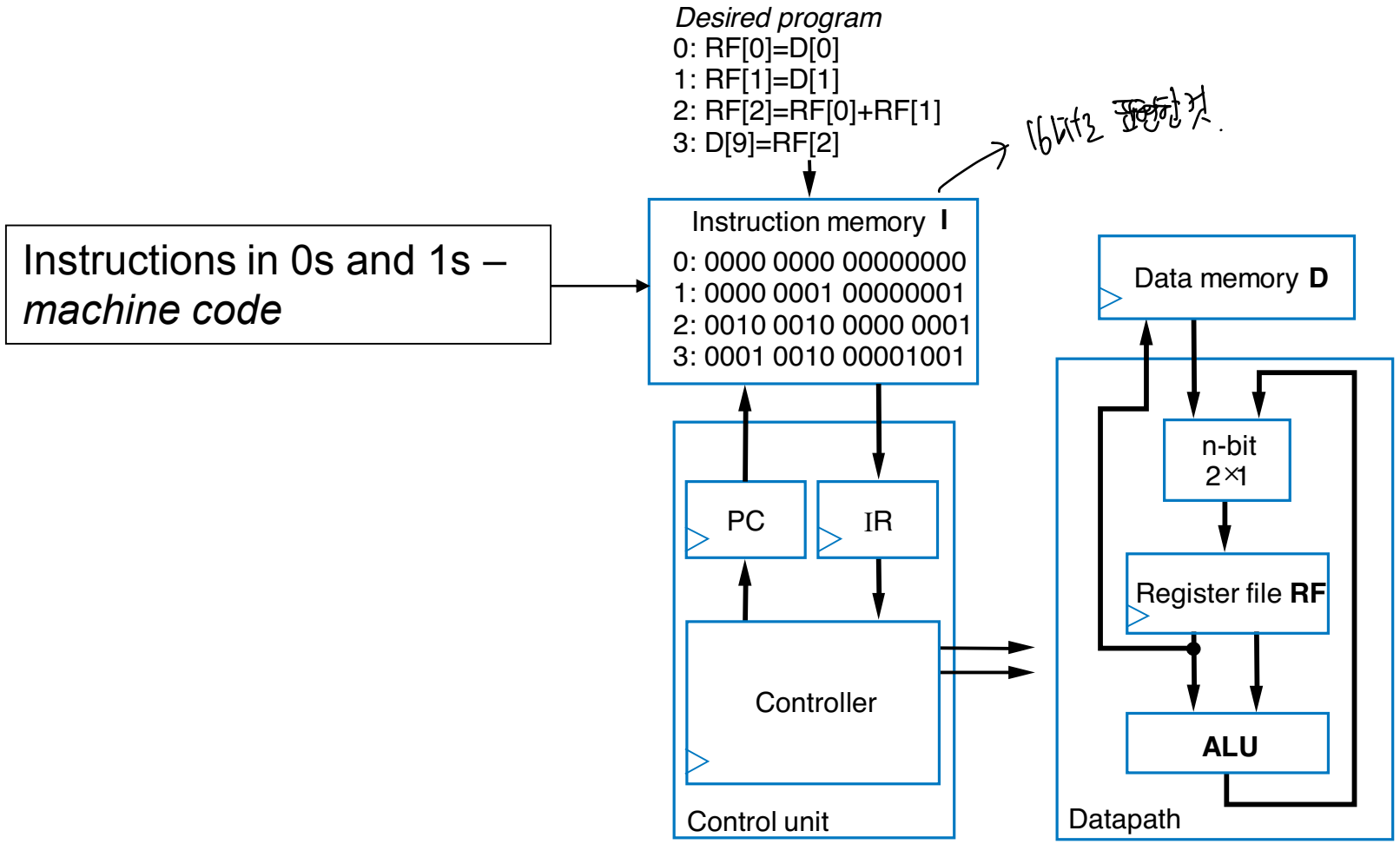


Instruction	CPI
Load	5 cycles = Fetch (2) + Decode (1) + Execute (2)
Store	4 cycles = Fetch (2) + Decode (1) + Execute (1)
Add	4 cycles = Fetch (2) + Decode (1) + Execute (1)
Mult	4 cycles = Fetch (2) + Decode (1) + Execute (1)

↳ 이것으로 전체 cycle을 구할 수 있음 ⇒ critical path 계산가능
⇒ 동작주파수도 알게됨.

Four-Instruction Processor Example:
Assembly Program for Four-Instruction Processor

- Example Program in machine code



Four-Instruction Processor Example:
Assembly Program for Four-Instruction Processor

- Machine code (0s and 1s) hard to work with
- Assembly code – Uses mnemonics (니모닉)
  - ***Load*** instruction—**LD Ra, d**
    - specifies the operation *RF[a]=D[d]*.
  - ***Store*** instruction—**ST d, Ra**
    - specifies the operation *D[d]=RF[a]*
  - ***Add*** or ***MULT*** instruction—**ADD** or **MULT Ra, Rb, Rc**
    - specifies the operation *RF[a]=RF[b]+RF[c] or RF[a]=RF[b]*RF[c]*

기계어는 아래와 같이 어셈블리코드로 변환.

Desired program

0: RF[0]=D[0]
1: RF[1]=D[1]
2: RF[2]=RF[0]+RF[1]
3: D[9]=RF[2]

machine code	assembly code
0: 0000 0000 00000000	0: LD R0, 0
1: 0000 0001 00000001	1: LD R1, 1
2: 0010 0010 0000 0001	2: ADD R2, R0, R1
3: 0001 0010 00001001	3: ST 9, R2

machine code ⟸ (어셈블러) assembly code

Four-Instruction Processor Example:
Performance Evaluation – 3-tap FIR filter

S·W로 구현했을 때

- Software Approach Using Four-Instruction Processor

↳ 명령에서 쓰는 방법.

C-Code 3-tap FIR filter

```
extern short x[102];
extern short c[3];
int main()
{
  short i, y[100];
  for(i =2;i<102;i++)
     y[i-2] = c[0]*x[i] +c[1]*x[i-1] + c[2]*x[i-2];
  return 0;
}
```

Assembly Code

```
0:   LD R0, 0 //RF[0]=c[0]        5 cycles
1:   LD R1, 2 //RF[1]=c[1]        5 cycles
2:   LD R2, 4 //RF[2]=c[2]        5 cycles
                                  } 15 cycles

3.   LD R3, 6 //RF[3]=x[0]        5 cycles
4:   LD R4, 8 //RF[4]=x[1]        5 cycles
5:   LD R5, 10 //RF[5]=x[2]       5 cycles
6:   MULT R3, R3, R2              4 cycles
7:   MULT R4, R4, R1              4 cycles
8:   MULT R5, R5, R0              4 cycles
9:   ADD R4, R4, R3               4 cycles
10:  ADD R5, R5, R4               4 cycles
11:  ST 300, R5 //y[0]=RF[5]      4 cycles
                                  } 39 cycles
                                  +
                                  +
                                  :
                                  +
894: LD R3, 204 //RF[3]=x[99]
895: LD R4, 206 //RF[4]=x[100]
896: LD R5, 208 //RF[5]=x[101]
897: MULT R3, R3, R2
898: MULT R4, R4, R1
899: MULT R5, R5, R0
900: ADD R4, R4, R3
901: ADD R5, R5, R4
902: ST 498, R5 //y[99]=RF[5]
                                  } 39 cycles
```

for문 100번 = 100 times

Loop Body

- Total Cycle Counts

  = 15 + 39 x 100 = 3915 cycles

동작 주파수 : 100 MHz

Four-Instruction Processor Example:
Performance Evaluation – 3-tap FIR filter

H.W

- Hardware Approach Using Custom Processor

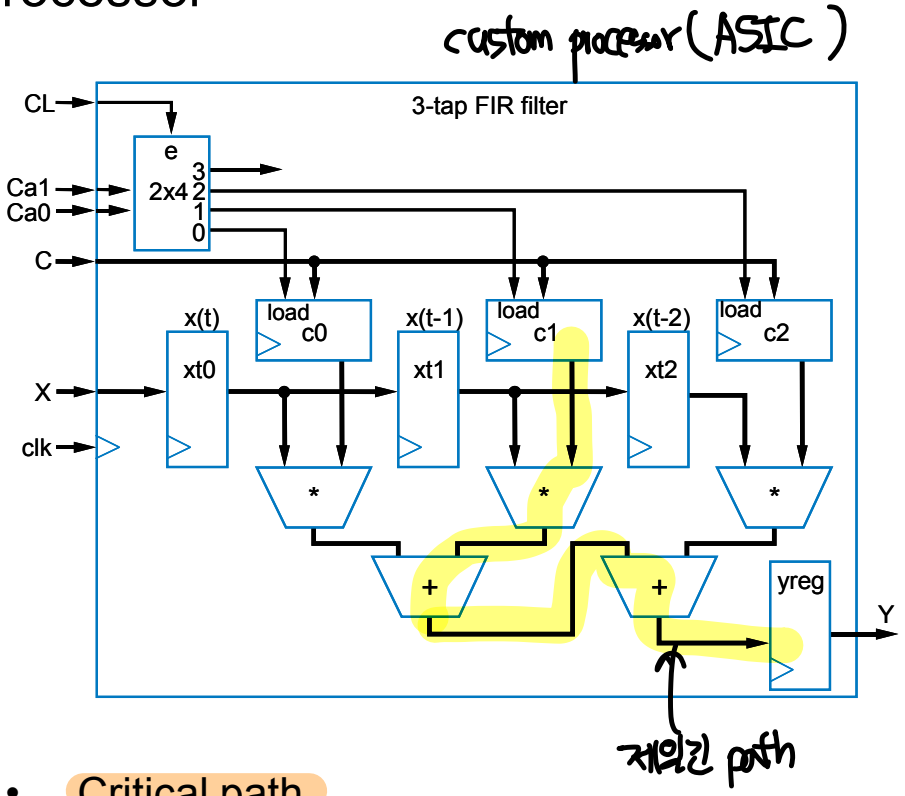
<여긴 제대로 설명X>

- Loading 3 Constants from Data Memory **D**
  - 1(Load from D) + 1(Load to C0) +
    1(Load from D) + 1(Load to C1) +
    1(Load from D) + 1(Load to C2) = 6 Cycles

- The first valid result from Y to D
  - 1(Load from D) + 1(Load to xt0) +
    1(Move to xt1) + 1(Move to xt2) +
    1(result to yreg) + 1 (Store to D) = 6 Cycles

- Successive load operations are ongoing during the remained 99 cycles.

- Therefore, Total Cycle Count = 6 + 6 + 99 = 111 cycles



- Critical path
  - Multiplier (8 ns) + Adder (2ns) + Adder (2ns) = 12 ns
- Operating Frequency: 83 MHz

Four-Instruction Processor Example:
Performance Evaluation – 3-tap FIR filter

- Performance Comparison between Hardware and Software

H·W와 S·W 성능비교.

Analysis / Type	Cycle Counts	Critical Path Delay /Operating Frequency	전체 수행시간 **Execution Time** (Cycle Count x Critical Path Delay)	**Speed-Up** (SW Ex' Time / HW EX' Time)
Software	3915	10 ns / 100 MHz	3915 x 10 ns = 39150 ns	-
Hardware	111	12 ns / 83 MHz	111 x 12 ns = 1332 ns	**29.4 X Faster** (= 39150 ns / 1332 ns)

S.W는 왜 H.W 보다 느린가?

CLK을 키워서 전력소모가 큼. → 유연성 가지고있음. S.W의 한계.

S.W는 수 많은 명령어에 대한 fetch. decode, excute 를 진행하기 때문에. 과정 ↑↑

H.W는 특정기능을 처리하기 위해 불필요한 과정이 없음. → 효율적 ↑↑, 전력소모량이 적음.

↳ 모든 기능을 처리할 수 있도록 함께.

CLK을 높이는 방식이 아닌 병렬성으로 성능개선 유리.