**1. ISA**

Addi rd = rt+immediate

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Op(000) | | | rt | | | rd | | | immediate | | | | | | |

Subi rd = rt-immediate

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Op(001) | | | rt | | | rd | | | Immediate | | | | | | |

Sw (rt+immediate)주소가 가리키는 곳에 rd값을 넣는다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Op(010) | | | rt | | | rd | | | Immediate | | | | | | |

Lw (rt+immediate)주소가 가리키는 곳의 값을 rd에 넣는다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Op(011) | | | rt | | | rd | | | immediate | | | | | | |

Beq (rt-rd)가 0이면 pc+immediate로 명령어를 이동시킨다

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Op(100) | | | rt | | | rd | | | immediate | | | | | | |

Sll rd에 (rt<<shamt)를 넣는다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Op(101) | | | rt | | | rd | | | Shamt(immediate) | | | | | | |

**2. STAGE**

1.IF: Instruction fetch

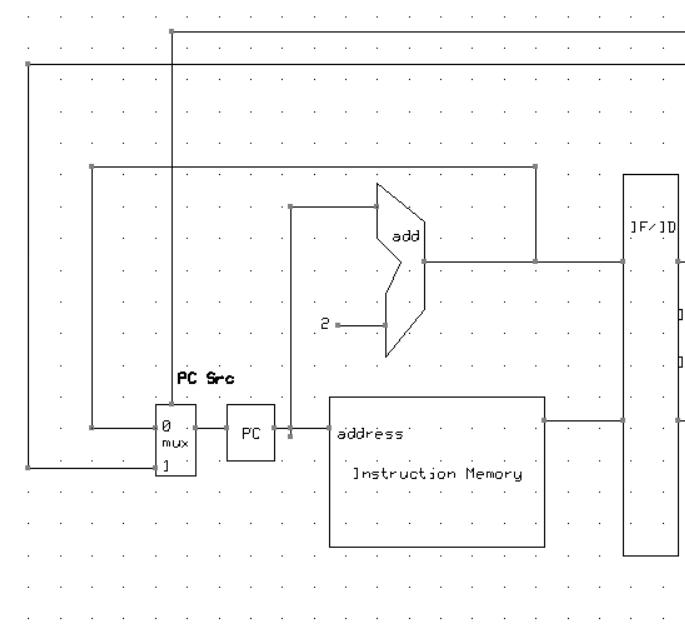
2.ID: Instruction decoding

3.EX: Execute instruction

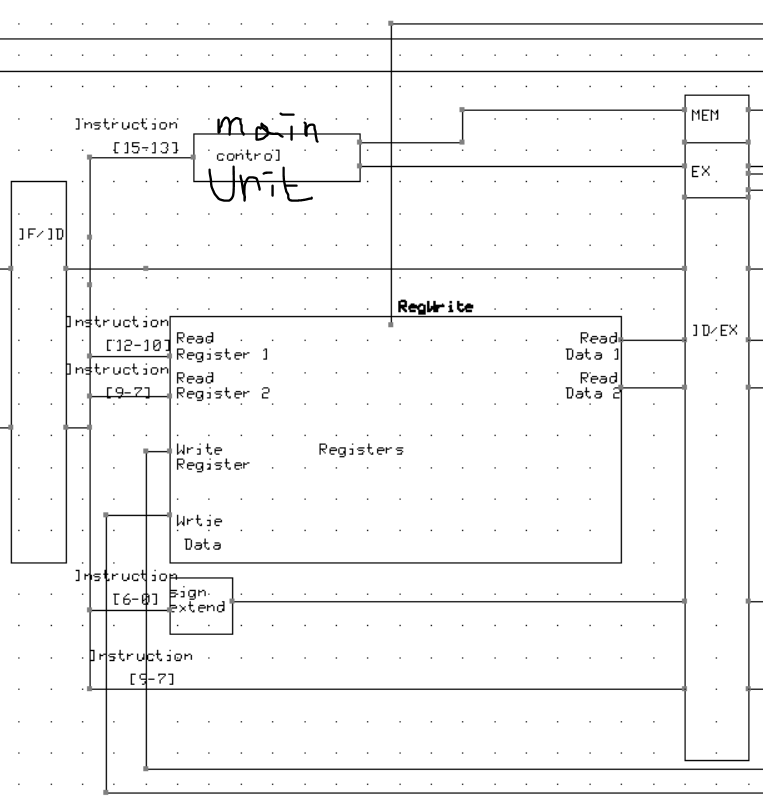
4.MEM: Memory access and write back to register

**3. DESIGN**

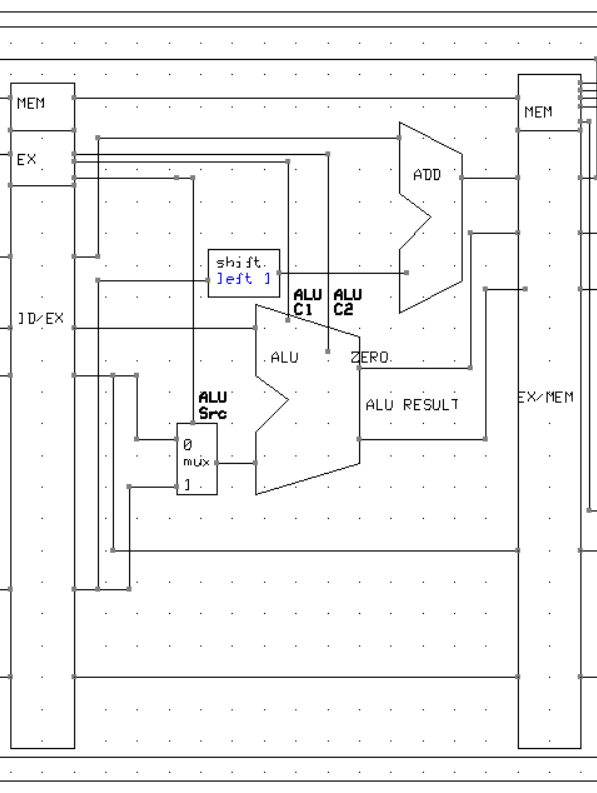
Instruction fetch step



Instruction decoding step



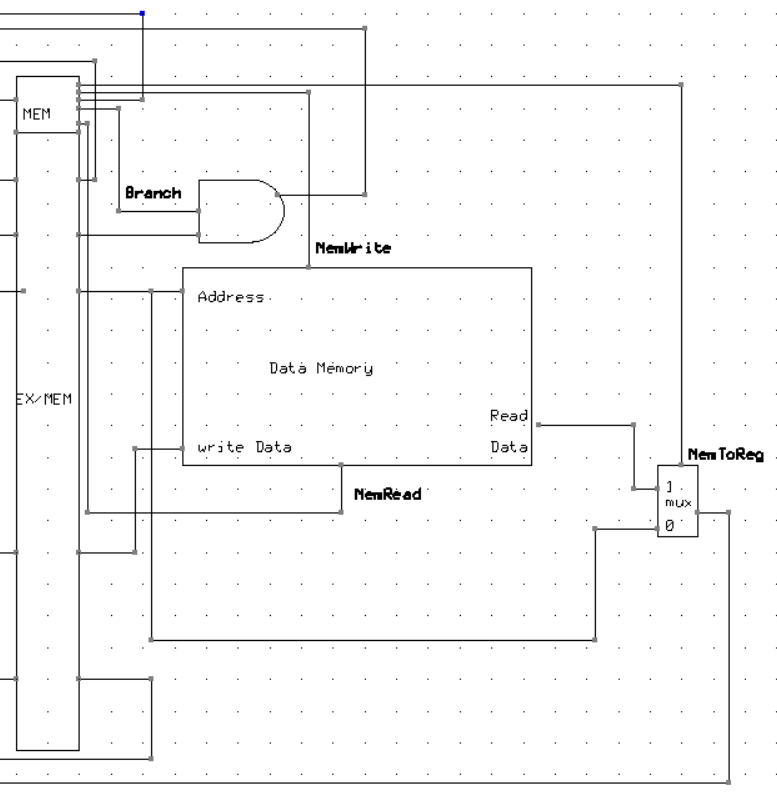
Execution Instruction step



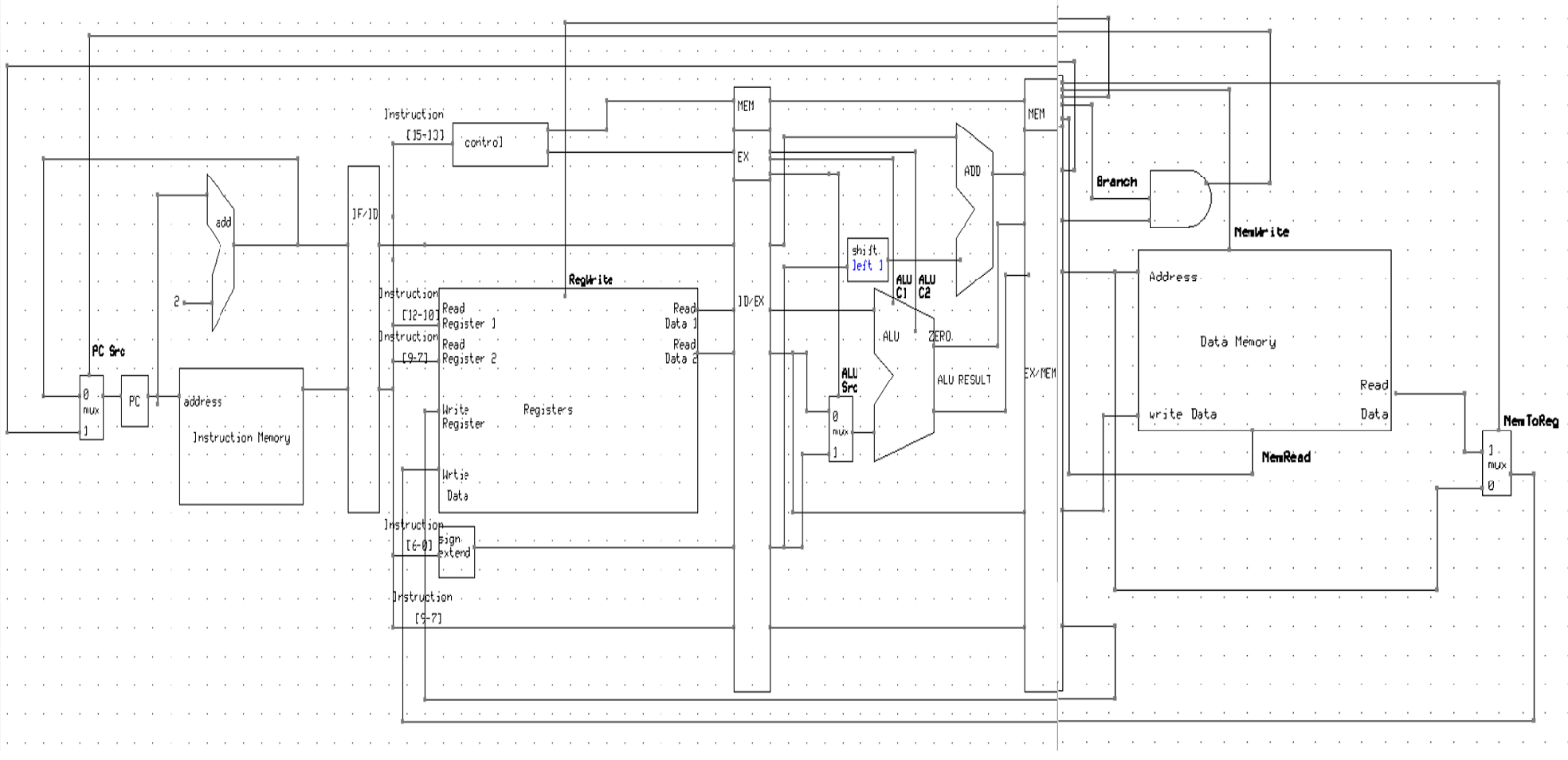
ZERO는 ALU에서 빼기 연산을 할 때 결과가 0이면 1을 출력하고 0이 아니면 0을 출력하게 됨.

|  |  |  |
| --- | --- | --- |
| ALU C1의 값 | ALU C2의 값 | ALU가 취하는 행동 |
| 0 | 0 | 덧셈 |
| 0 | 1 | 뺄셈 |
| 1 | 0 | 왼쪽 쉬프트 연산 |

Memory access and write back to register step



전체 구조

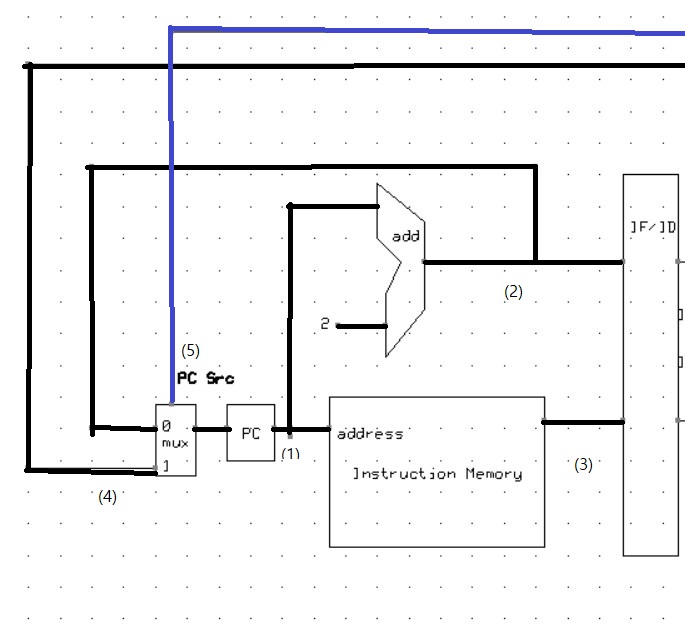


**4. Main Control Unit 제어신호**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | RegWrite | ALUSrc | ALUC1 | ALUC2 | Branch | MemWrite | MemRead | MemToReg |
| Addi | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Subi | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| Sw | 0 | 1 | 0 | 0 | 0 | 1 | 0 | X |
| Lw | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| Beq | 0 | 0 | 0 | 1 | 1 | 0 | 0 | X |
| sll | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

**5.신호 흐름**

1.IF(Instruction fetch, 1step): 모든 명령어 공통



(1)에는 PC의 값이 나오게 되며 이 값은 현재 명령어의 주소와 같습니다.

(2)에는 PC+2의 값이 나오게 되며 2바이트를 1워드라고 할 때 이는 다음 명령어의 주소와 같습니다.

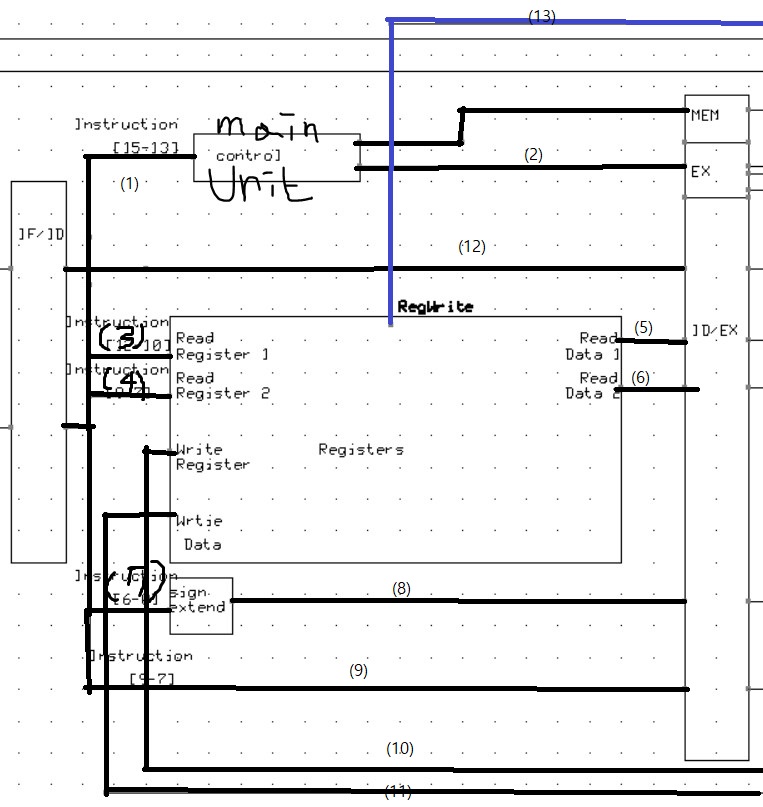
(3)은 명령어가 나오게 됩니다.

(4)는 Memory access step(4step)이후에 나오는 값으로 beq에서 조건이 맞을 시에 branch할 주소를 나타냅니다.

(5)는 mutex를 컨트롤하는 PC Src신호로 이 신호는 Memory access step(4step) 이후에 나오게 됩니다. 이 신호가 0이면 mutex는 다음 명령어 주소를 보내게 되고 1이면 beq 에서 가리키는 branch 주소를 보내게 됩니다.

이러한 과정을 거쳐서 명령어를 가져오게 됩니다.

2.ID(Instruction Decoding, 2step): 모든 명령어 공통



(1)에는 명령어 중에 op(opcode)가 나오게 됩니다.

(2)에는 opcode를 해석해서 나온 control신호들입니다. Main Control 제어 유닛 신호에 해당합니다.

(3)에는 명령어에 있는 rt의 번호가 나오게 됩니다.

(4)에는 명령어에 있는 rd의 번호가 나오게 됩니다.

(5)에는 rt가 가리키는 레지스터의 값이 나오게 됩니다.

(6)에는 rd가 가리키는 레지스터의 값이 나오게 됩니다.

(7)에는 명령어에 있는 immediate(shamt)값이 나오게 됩니다.

(8)에는 immediate의 sign-extended 된 값이 나오게 됩니다. 이 때 이 값들은 16비트로 이루어져 있습니다.

(9)는 memory access(4step)에 쓰이는 값들입니다. Rd의 번호가 나오게 됩니다.

(10)은 쓸 레지스터의 번호가 나옵니다. Memory access(4step)이후에 나오게 됩니다.

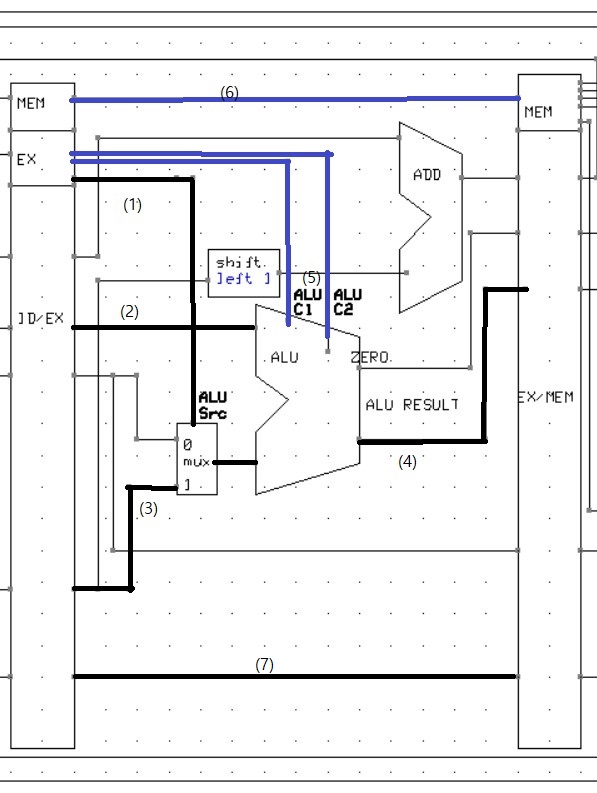
(11)은 쓸 레지스터의 값이 나옵니다. Memory access(4step)이후에 나오게 됩니다.

(12)는 instruction fetch(1step)에 PC+2의 값을 의미합니다.

(13)은 RegWrite로 이 값에 따라 레지스터를 쓸 지 여부를 결정합니다.

이 과정을 통해 명령어를 해석하게 됩니다.

3.EX(Execute instruction, 3step): Addi,Subi, Sll



(1)은 ALU Src의 컨트롤 신호로 어떤 것을 ALU source로 받아들일지 결정하는 것입니다. Addi, Subi, sll는 rt, immediate(shamt)를 사용하게 되므로 이 컨트롤 신호는 1이어야 합니다. 그래서 immediate를 받아들이게 됩니다.

(2)는 instruction decoding(2step)에서 나온 rt의 레지스터 값입니다.

(3)은 instruction decoding(2step)에서 나온 immediate(shamt)의 값입니다.

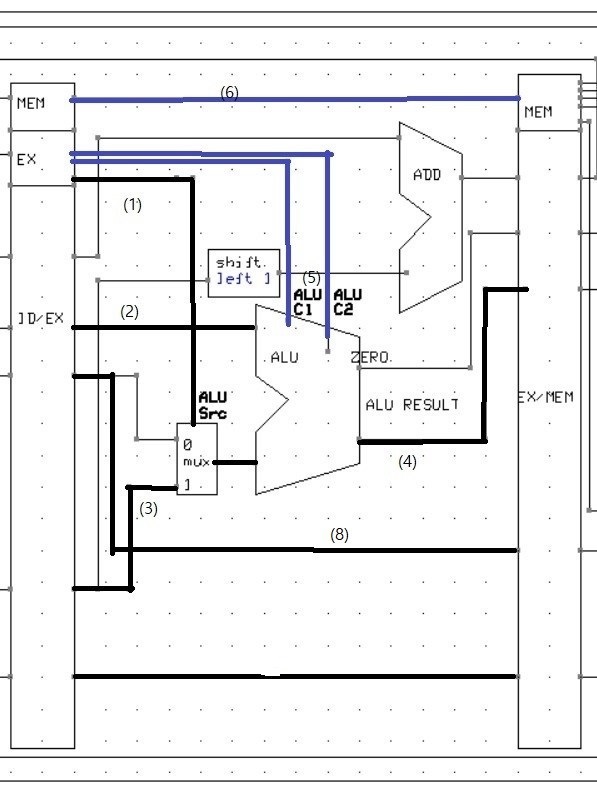
(4)는 계산 결과값입니다. 명령어가 addi일 경우 rt+immediate 명령어가 subi일 경우 rt-immediate가 sll일 경우 (rt<<shamt)가 나오게 됩니다.

(5)는 ALU 컨트롤 신호입니다. 이 값에 따라서 ALU가 덧셈, 뺄셈, 쉬프트 연산을 하게 됩니다. Addi는 (ALU C1, ALU C2) = (0, 0) subi는 (ALU C1, ALU C2) = (0,1), sll는 (ALU C1, ALU C2) = (1,0)이 들어가게 됩니다.

(6)은 memory access(4step)에 쓰일 컨트롤 신호입니다.

(7)은 instruction decoding(2step)에 나온 rd의 값입니다. Addi, subi, sll의 경우 연산된 결과를 rd번 레지스터에 집어넣습니다.

4.EX(Execute instruction, 3step): Sw



(1) sw같은 경우 immediate를 ALU source로 받아들여야 합니다. 따라서 이 신호는 1이어야 합니다.

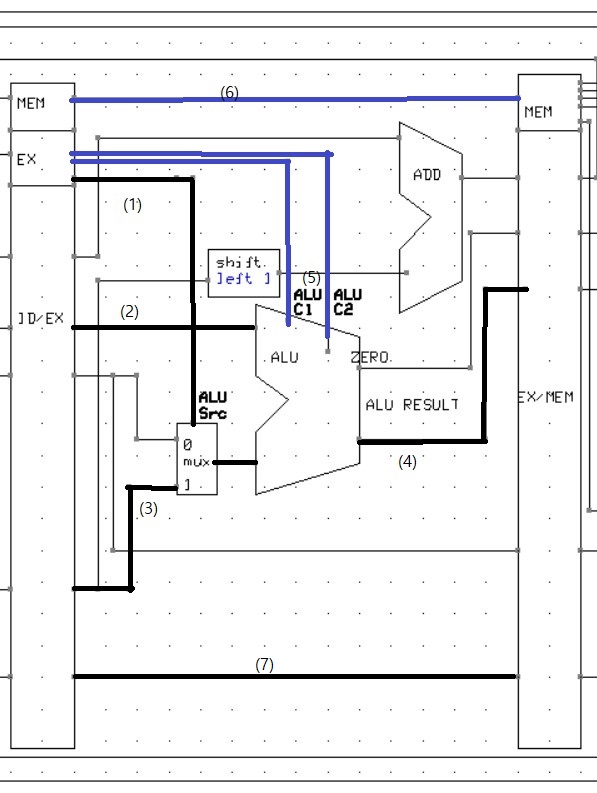
(2)~(3) 이는 execution instruction step(3step)의 addi, subi, sll와 같습니다.

(4)rt+immediate의 값이 나오게 됩니다. 이 때 이 값은 쓸 메모리의 주소입니다.

(5) 덧셈 연산을 수행해야 합니다. 따라서 (ALU C1, ALU C2) = (0, 0)입니다.

(8)은 rd의 값입니다. 이 값이 메모리에 저장될 값입니다.

5.EX(Execute instruction, 3step): Lw



(1) lw같은 경우 immediate를 ALU source로 받아들여야 합니다. 따라서 이 신호는 1이어야 합니다.

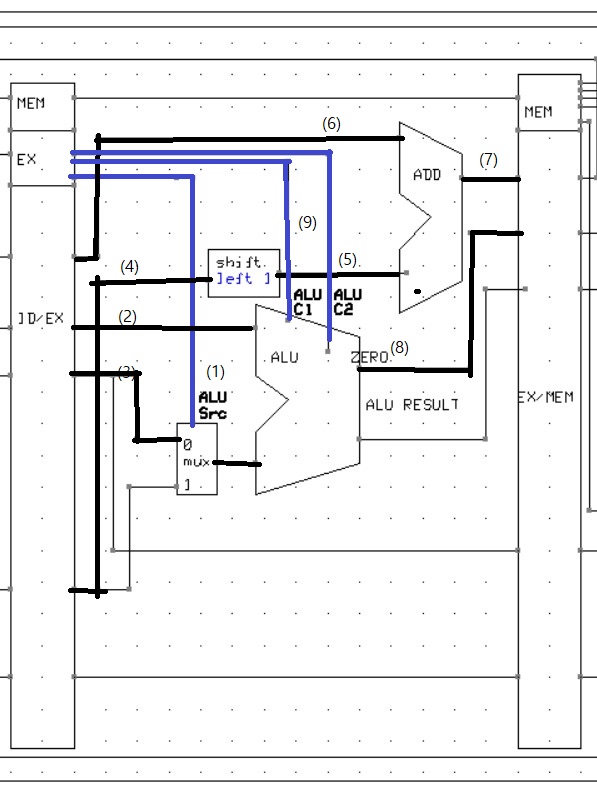
(2)~(3) 이는 execution instruction step(3step)의 addi, subi, sll와 같습니다.

(4)rt+immediate의 값이 나오게 됩니다. 이 때 이 값은 읽어올 메모리의 주소입니다.

(5) 덧셈 연산을 수행해야 합니다. 따라서 (ALU C1, ALU C2) = (0, 0)입니다.

(7)은 instruction decoding(2step)에 나온 rd의 값입니다. lw의 경우 메모리의 값을 rd번 레지스터에 저장하게 됩니다.

6.EX(Execute instruction, 3step): Beq



(1) ALU Source는 rd이어야 합니다 따라서 신호는 0이어야 합니다.

(2)는 instruction decoding(2step)에서 나온 rt의 레지스터 값입니다.

(3)는 instruction decoding(2step)에서 나온 rd의 레지스터 값입니다.

(4)은 instruction decoding(2step)에서 나온 immediate의 값입니다.

(5)는 immediate의 값을 왼쪽으로 shift 1번 한 값입니다. 1워드의 크기가 2바이트이므로 shift left 1장치를 이용하였습니다.

(6)은 instruction decoding(2step)에서 나온 PC+2입니다.

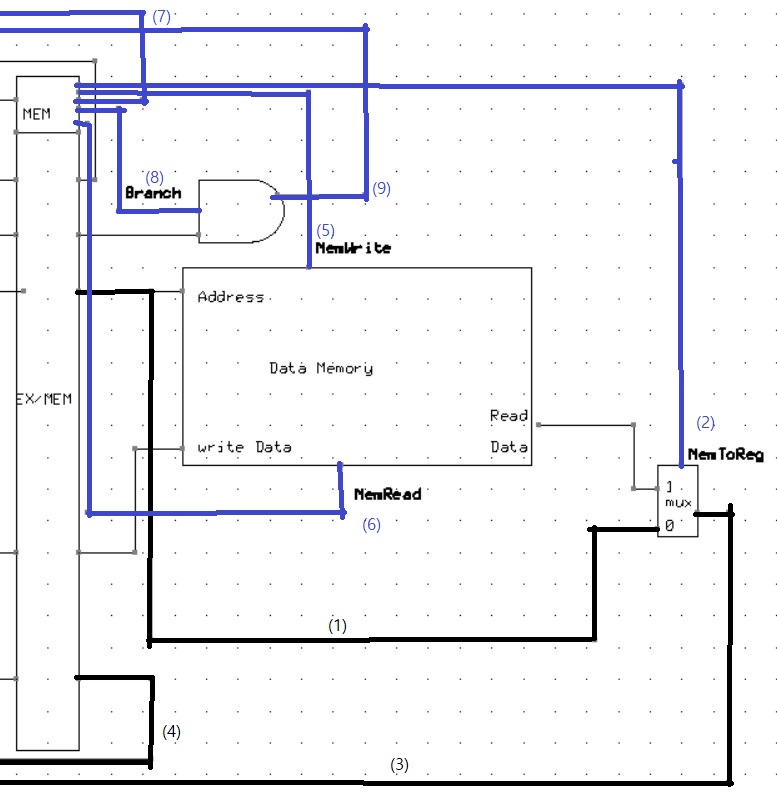
(7)은 (PC+2+immediate) 즉 beq가 가리키는 branch주소 값입니다.

(8)은 rt-rd의 값이 0일 때 1이 되게 됩니다. 이 값을 이용하여 분기를 할지 말지 정하게 됩니다.

(9)는 빼기 연산을 수행해야 하므로 (ALU C1, ALU C2) = (0,1)입니다.

이 과정을 통해 명령어를 수행하게 됩니다.

7.MEM(Memory Access And Write Back, 4step): Addi, Subi, Sll



(1) execution(3step)에서 나온 ALU연산 값입니다.

(2) 이 신호는 메모리에서 Register로 값을 쓸건지 여부를 말해줍니다. Addi, subi, sll은 메모리의 값을 필요치 않으므로 이 신호는 0이어야 합니다.

(3) instruction decoding(2step)으로 돌아갈 값으로 쓸 레지스터 값을 의미합니다. 여기에선 addi, subi, sll을 연산하면서 나온 값이 나오게 됩니다

(4) instruction decoding(2step)으로 돌아갈 값으로 쓸 레지스터의 번호를 의미합니다. 여기에서는 rd가 나오게 됩니다.

(5) 메모리에 쓸 건지 여부를 물어보는 신호입니다. 안 쓸 것이므로 0이 들어갑니다.

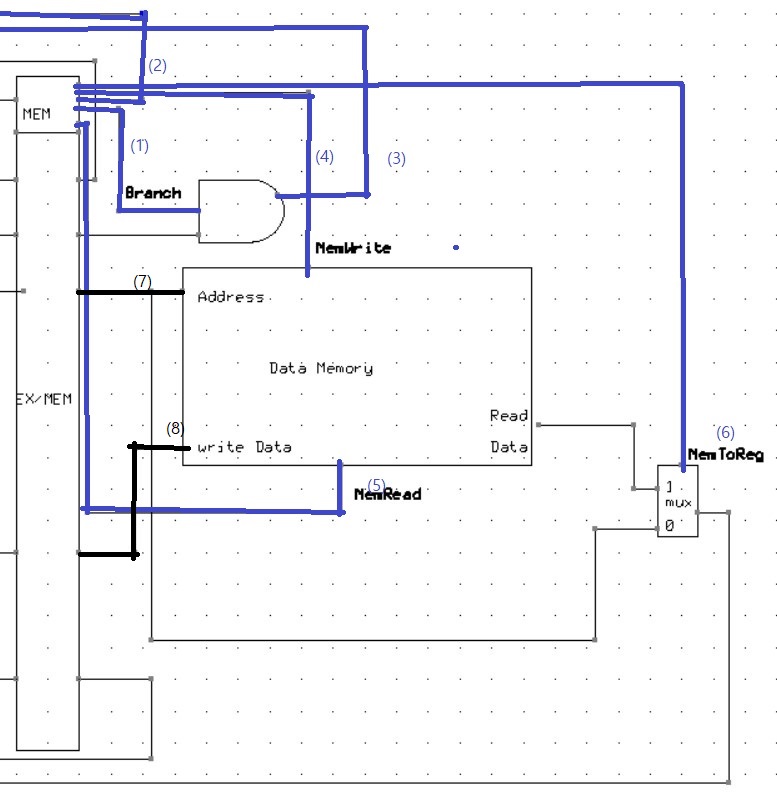
(6) 메모리를 읽을 것인지 여부를 물어보는 신호입니다. 안 읽을 것이므로 0이 들어갑니다.

(7) RegWrite를 의미하며 instruction decoding(2step)에 있는 레지스터에 값을 쓸 건지 여부를 물어보는 신호입니다. 값을 써야 하므로 1이 들어갑니다.

(8) branch 할 건지 여부를 물어보는 신호입니다. Branch를 안 할 것이므로 0이 들어갑니다.

(9) PC Src값으로 instruction fetch(1step)으로 돌아가게 됩니다. 여기에서는 무조건 0이 나옵니다.

8.MEM(Memory Access And Write Back, 4step): Sw



(1) branch 할 건지 여부를 물어보는 신호입니다. Branch를 안 할 것이므로 0이 들어갑니다.

(2) RegWrite를 의미하며 instruction decoding(2step)에 있는 레지스터에 값을 쓸 건지 여부를 물어보는 신호입니다. 값을 안 쓸 것이므로 0이 들어갑니다.

(3) PC Src값으로 instruction fetch(1step)으로 돌아가게 됩니다. 여기에서는 무조건 0이 나옵니다.

(4) 메모리에 쓸 건지 여부를 물어보는 신호입니다. 쓸 것이므로 1이 들어갑니다.

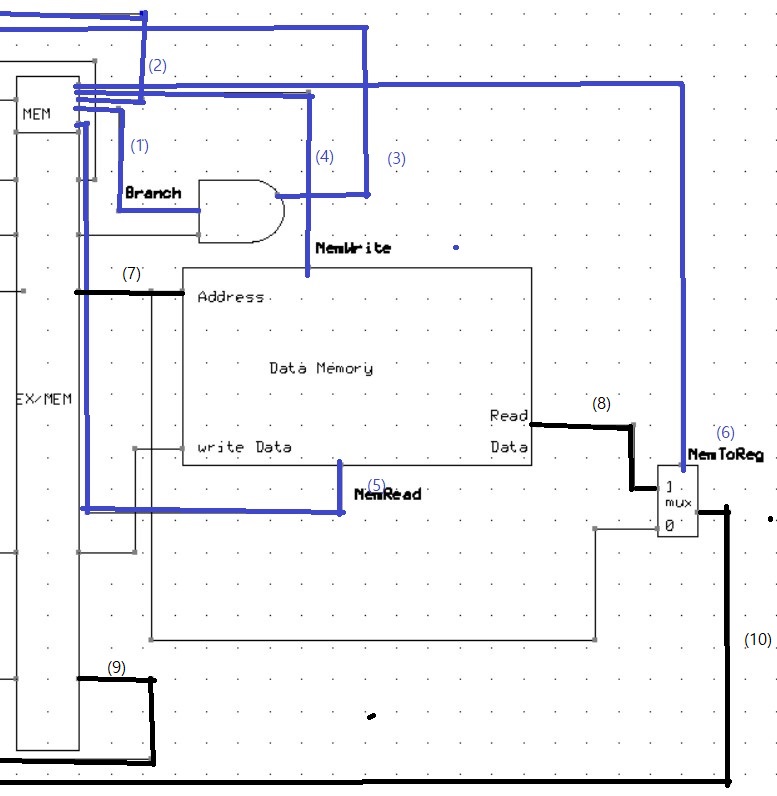
(5) 메모리를 읽을 것인지 여부를 물어보는 신호입니다. 안 읽을 것이므로 0이 들어갑니다.

(6) 이 신호는 메모리에서 Register로 값을 쓸건지 여부를 말해줍니다. Sw는 어떤 것이든 상관 없습니다.

(7)은 execution step(3step)에서 계산된 결과입니다. 메모리에서 써야 할 곳의 주소를 의미합니다.

(8)은 rd의 값이며 이 값을 메모리에 쓰게 됩니다.

9.MEM(Memory Access And Write Back, 4step): lw



(1) branch 할 건지 여부를 물어보는 신호입니다. Branch를 안 할 것이므로 0이 들어갑니다.

(2) RegWrite를 의미하며 instruction decoding(2step)에 있는 레지스터에 값을 쓸 건지 여부를 물어보는 신호입니다. 값을 쓸 것이므로 1이 들어갑니다.

(3) PC Src값으로 instruction fetch(1step)으로 돌아가게 됩니다. 여기에서는 무조건 0이 나옵니다.

(4) 메모리에 쓸 건지 여부를 물어보는 신호입니다. 안 쓸 것이므로 0이 들어갑니다.

(5) 메모리를 읽을 것인지 여부를 물어보는 신호입니다. 읽을 것이므로 1이 들어갑니다.

(6) 이 신호는 메모리에서 Register로 값을 쓸건지 여부를 말해줍니다. 메모리의 값을 써야 하므로 1이 들어가야 합니다.

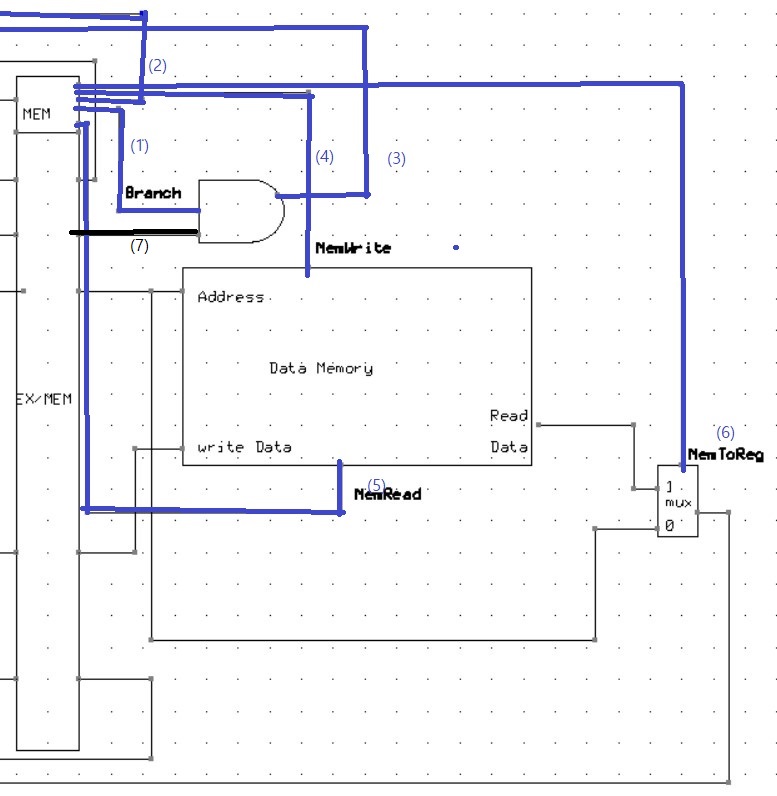
(7)은 execution step(3step)에서 계산된 결과입니다. 메모리에서 읽어야 할 곳의 주소를 의미합니다.

(8)은 메모리에서 읽어온 값을 의미합니다.

(9) instruction decoding(2step)으로 돌아갈 값으로 쓸 레지스터의 번호를 의미합니다. 여기에서는 rd가 나오게 됩니다.

(10) instruction decoding(2step)으로 돌아갈 값으로 쓸 레지스터 값을 의미합니다. 여기에선 읽어온 메모리의 값이 나오게 됩니다.

10.MEM(Memory Access And Write Back, 4step): beq



(1) branch 할 건지 여부를 물어보는 신호입니다. Branch를 할 것이므로 1이 들어갑니다.

(2) RegWrite를 의미하며 instruction decoding(2step)에 있는 레지스터에 값을 쓸 건지 여부를 물어보는 신호입니다. 값을 안 쓸 것이므로 0이 들어갑니다.

(3) PC Src값으로 instruction fetch(1step)으로 돌아가게 됩니다. 여기에서는 zero신호가 1이면 1이 나오고 zero신호가 0이면 0이 나오게 되어 있습니다.

(4) 메모리에 쓸 건지 여부를 물어보는 신호입니다. 안 쓸 것이므로 0이 들어갑니다.

(5) 메모리를 읽을 것인지 여부를 물어보는 신호입니다. 안 읽을 것이므로 0이 들어갑니다.

(6) 이 신호는 메모리에서 Register로 값을 쓸 건지 여부를 말해줍니다. 여기에서는 상관 없습니다.

(7)은 zero 신호입니다. 이 값이 1이면 분기를 하게 됩니다.

이 과정을 통해서 메모리에 읽기 쓰기를 하게 되고 다시 레지스터에 쓰게 됩니다.