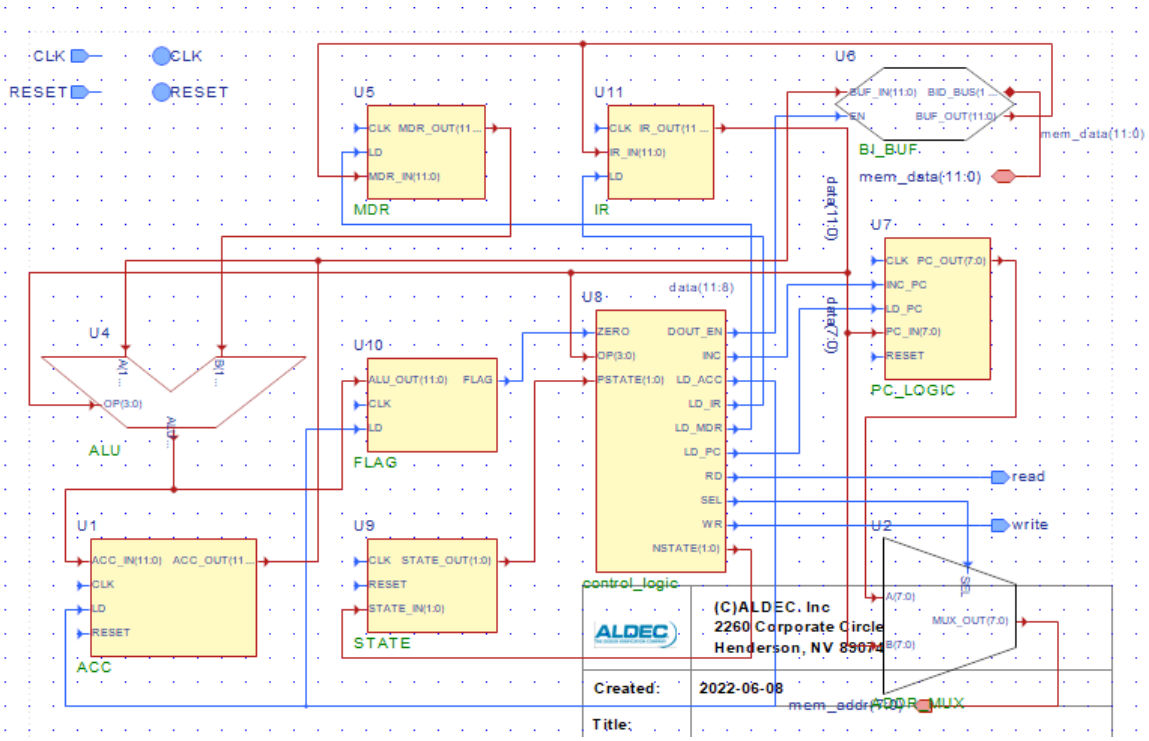
마이크로 프로세서
내가 설계한 CPU 프로그램 설명서

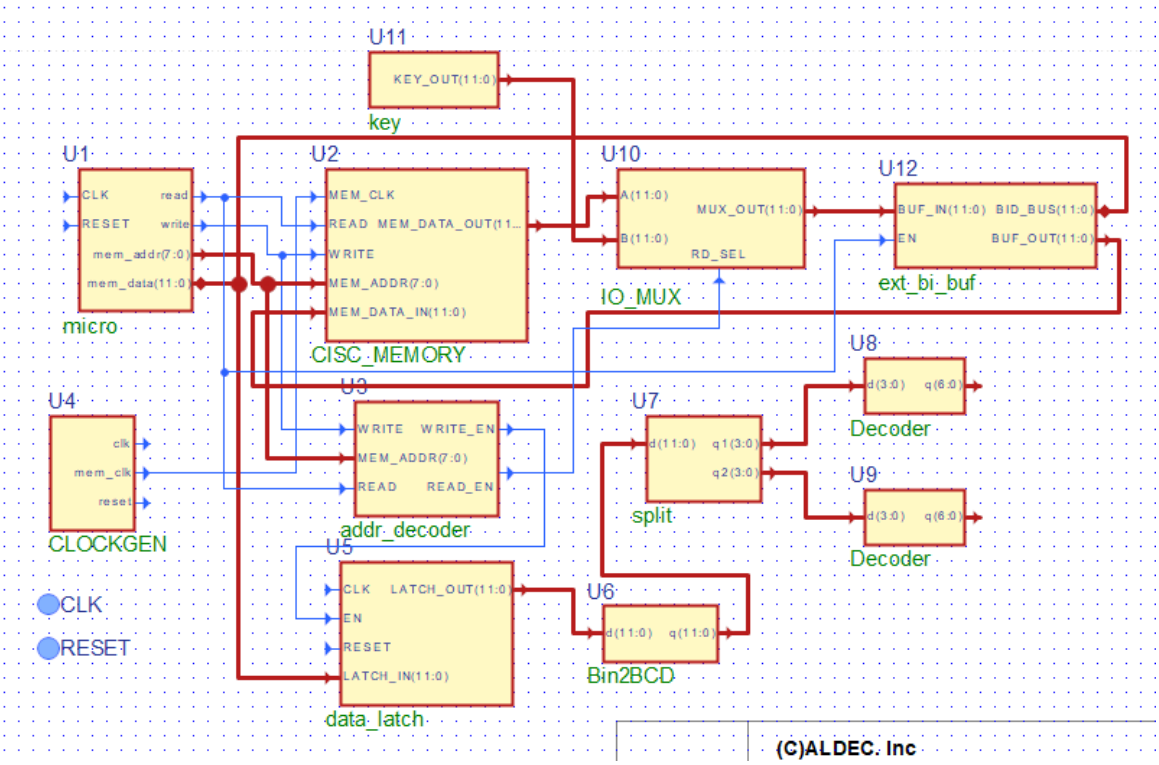
〈목차〉

설계일자 : 2022.12.07
수업명 : 마이크로 프로세서
학번 : 201807558
이름 : 김도훈

1. 프로그램을 설계한 과정



최종적으로 제가 설계한 CPU의 bde 입니다.
저는 이 CPU로 이번엔 구구단을 구현하고자 하였습니다.



그리고 위 bde는 7_Segment의 bde 입니다.

2. CPU의 프로그램 설명 – 설명시작

```
31  entity CISC_MEMORY is
32       port(
33           MEM_CLK : in STD_LOGIC;
34           READ : in STD_LOGIC;
35           WRITE : in STD_LOGIC;
36           MEM_ADDR : in STD_LOGIC_VECTOR(7 downto 0);
37           MEM_DATA_IN : in STD_LOGIC_VECTOR(11 downto 0);
38           MEM_DATA_OUT : out STD_LOGIC_VECTOR(11 downto 0)
39           );
40  end CISC_MEMORY;
41
42  architecture CISC_MEMORY of CISC_MEMORY is
43
44  subtype MEM_WORD is std_logic_vector(11 downto 0);
45  type MEM_TABLE is array(255 downto 0) of MEM_WORD;
46  signal cells : MEM_TABLE := (
47  "000000001001", --11111111 번지 : Xn      start!  -- 구구단..! Xn절 로 설정 | 11단 까지 가능
48  "000000000001", --11111110 번지 : n                  Segment 가 2개 라서 최대 99
49  "000000000001", --11111101 번지 : one
50  "000000001010", --11111100 번지 : Last_n = 10        -- Xn * 9 까지..!
51  "XXXXXXXXXXXX", --11111011 번지 :
52  "XXXXXXXXXXXX", --11111010 번지 : tmp
53  "XXXXXXXXXXXX", --11111001 번지 : I/O Address(Output)
54  "XXXXXXXXXXXX", --11111000 번지 : I/O Address(Input)
55  "XXXXXXXXXXXX", --
56  "XXXXXXXXXXXX", --
57  "XXXXXXXXXXXX", --
58  "XXXXXXXXXXXX", --
59  "XXXXXXXXXXXX", --
60  "XXXXXXXXXXXX", --
```

```
285 "XXXXXXXXXXXX", --
286 "XXXXXXXXXXXX", --
287 "XXXXXXXXXXXX", --
288 "XXXXXXXXXXXX", --
289 "XXXXXXXXXXXX", --
290 "000000000000", --Halt
291 "011100000000", --Jump start    -- Polling end
292 "000100000000", --Skip_if_zero
293 "010011111100", --Xor Last_n
294 "011011111110", --Store n
295 "001011111101", --Add one
296 "010111111110", --Load n
297 "011011111001", --Store I/O_address(Output)
298 "010111111010", --Load temp
299 "011011111010", --Store temp
300 "100011111110", --Mul n (count)
301 "010111111111", --Load Xn
302 "010111111000");--start: Load I/O_address(Input) --Polling start
303 begin
304
305 process
306     variable address : integer range 0 to 255;
307     begin
308         wait until MEM_CLK'event and MEM_CLK='1';
309         address := conv_integer(unsigned(mem_addr));
310         if read='1' then
311             MEM_DATA_OUT <= cells(address);
312         elsif write ='1' then
313             cells(address) <= MEM_DATA_IN;
314         else
315         mem_data_out <= "ZZZZZZZZZZZZ";
316     end if;
317 end process;
318 end CISC_MEMORY;
```

2. CPU의 프로그램 설명 - 설명시작

==========

위 페이지의 사진은 cisc_memory 코드입니다.
저는 구구단 프로그램을 만들었고 위 코드는 9단을 출력하는 프로그램입니다.

제일 상위 “11111111” 번지를 Xn 이라 할 때 9값인 “00000001001” 를
넣었고 바로 다음 번지 “11111110” 번지를 n 이라하고 이것을 Xn * 1 부터
Xn * 9 까지의 카운트로 사용합니다.
그 다음 “11111101” 번지의 one 값은 카운트 n 을 1씩 증가 시켜주기 위해
똑같이 1을 넣어주었습니다.
그 다음 “11111100” 번지의 Last_n 값으로 카운트 n 이 그 값 까지만 증가하고
프로그램을 종료 할 수 있도록 값을 10(1010)로 지정하였습니다.
카운트 n 은 10 까지 늘어나지만 결과값은 *9 까지 출력됩니다.
그리고 “11111010” 번지의 tmp 는 “XXXXXXXXXXX” 으로 두는데
그 이유는 맨 아래의 MachineLanguage 가 연산되면서
그 값들을 tmp에 저장하기 위해서 입니다.
그리고 “11111001” 번지를 I/O Address (Output) 으로 정하고
“11111000” 번지를 I/O Address (Input) 으로 정합니다.

==========

이제 위 사진의 아래로 넘어가서 제가 설계한 MachineLanguage 를
해석하여 설명하겠습니다.
[** Load(0101), Store(0110), Add(0010), Xor(0100), Mul(1000)]
[** Skip-if-zero(0001), Halt(0000), Jump(0111)]

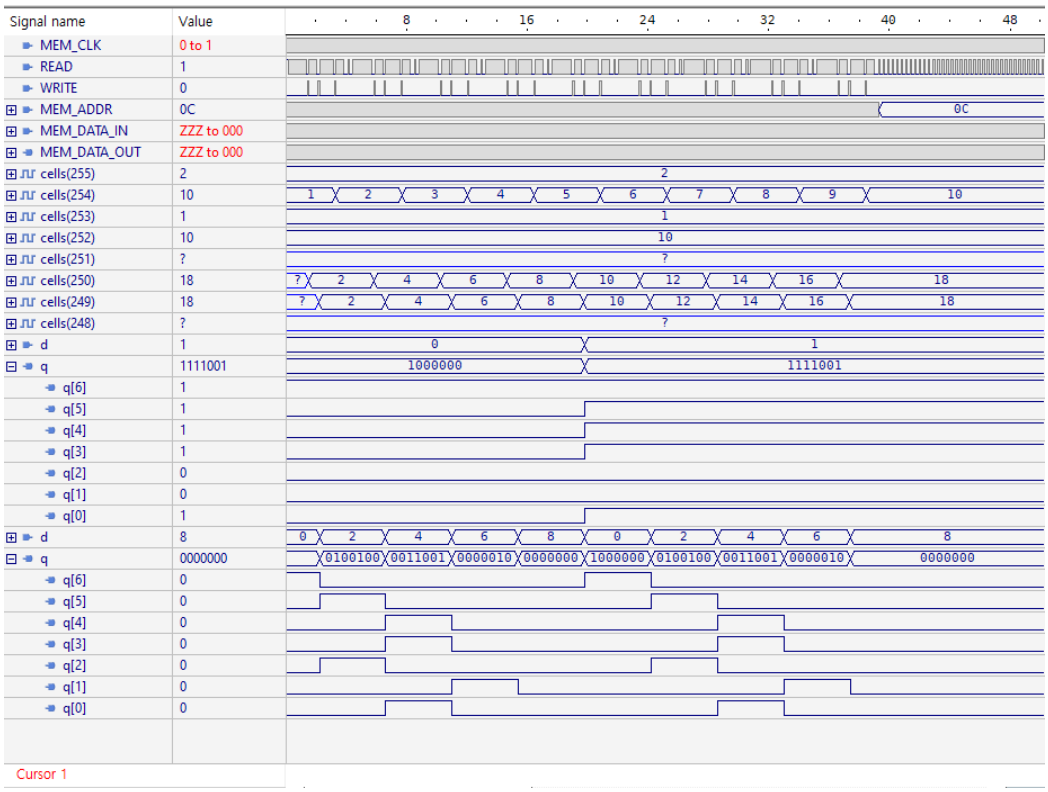
1. Load I/O Address(Input) 으로 “11111000” 번지의 값을 불러옵니다.
2. Load Xn 으로 제가 맨위 “11111111” 번지에서 정한 값(9)을 불러옵니다.
3. Mul n 으로 구구단의 앞 숫자 Xn 에 n 을 곱해줍니다.
4. Store temp 그 값을 temp 에 저장합니다.
5. Load temp 로 temp 값을 불러옵니다.
6. Store I/O Address(Output) 으로 “11111001” 번지에 출력값을 저장합니다.
7. Load n 으로 카운트 n 을 불러옵니다.
8. Add one 으로 카운트 n 에 1 만큼 더해줍니다.
9. Store n 으로 그 값을 다시 카운트 n 에 저장하여 갱신합니다.

10. Xor Last_n 으로 위에서 지정한 Last_n => 10(“1010”) 값과
    증가된 n 값을 Xor 연산을합니다.
11. Skip-if-zero 는 10번에서 Xor 값 zero flag의 값이 ‘1’ 일 경우 PC의 값을
    1 만큼 증가시켜서 다음 명령을 건너뛰게합니다.
    이 프로그램이 계속 실행되다가 n값이 Last_n과 같아진 상태로
    10번에서 Xor 연산을 하게되면 11번의 Skip-if-zero 가 다음 명령을
    건너 뛰게하여 Halt 명령을 만나 프로그램이 종료되도록 합니다.
12. Jump Start 는 카운트를 1부터 증가시키면서 Xor 연산을하고
    11번 Skip-if-zero 가 실행되지 않았을때 다시 Load Xn의 번지 값인
    “00000000” 으로 점프시켜줘서 프로그램을 지정한 값까지
    계속 실행하게 하는 명령어입니다.
13. Halt 는 11번에서 설명한 내용과 같이 프로그램을 종료시킵니다.

==========

2. CPU의 프로그램 설명 – 프로그램 실행 결과

프로그램은 바로 위의 내용이었던 목차 “설명시작” 의 내용대로 동작합니다.
저는 구구단 프로그램을 만들었고 메모리는 11비트까지 늘렸지만 아쉽게도
7 Segment 2개만 사용해서 2개로 표현 할 수 있는 최대치인 99 까지만
표현 가능하여 11단의 11*9 까지 결과를 볼 수 있습니다.



위 사진은 설명내용에 따라 실행 된 구구단 중 2단을 출력한 프로그램입니다.

아래 사진은 설명내용에 따라 실행 된 구구단 중 9단을 출력한 프로그램입니다.