(1a) SUB instruction:

- cycle 1: $IR = Mem[PC]$, $PC = PC + 4$
- cycle 2: $A = Mem[zero\text{-}ext(IR_{7-13})]$
- cycle 3: $B = Mem[zero\text{-}ext(IR_{0-6})]$

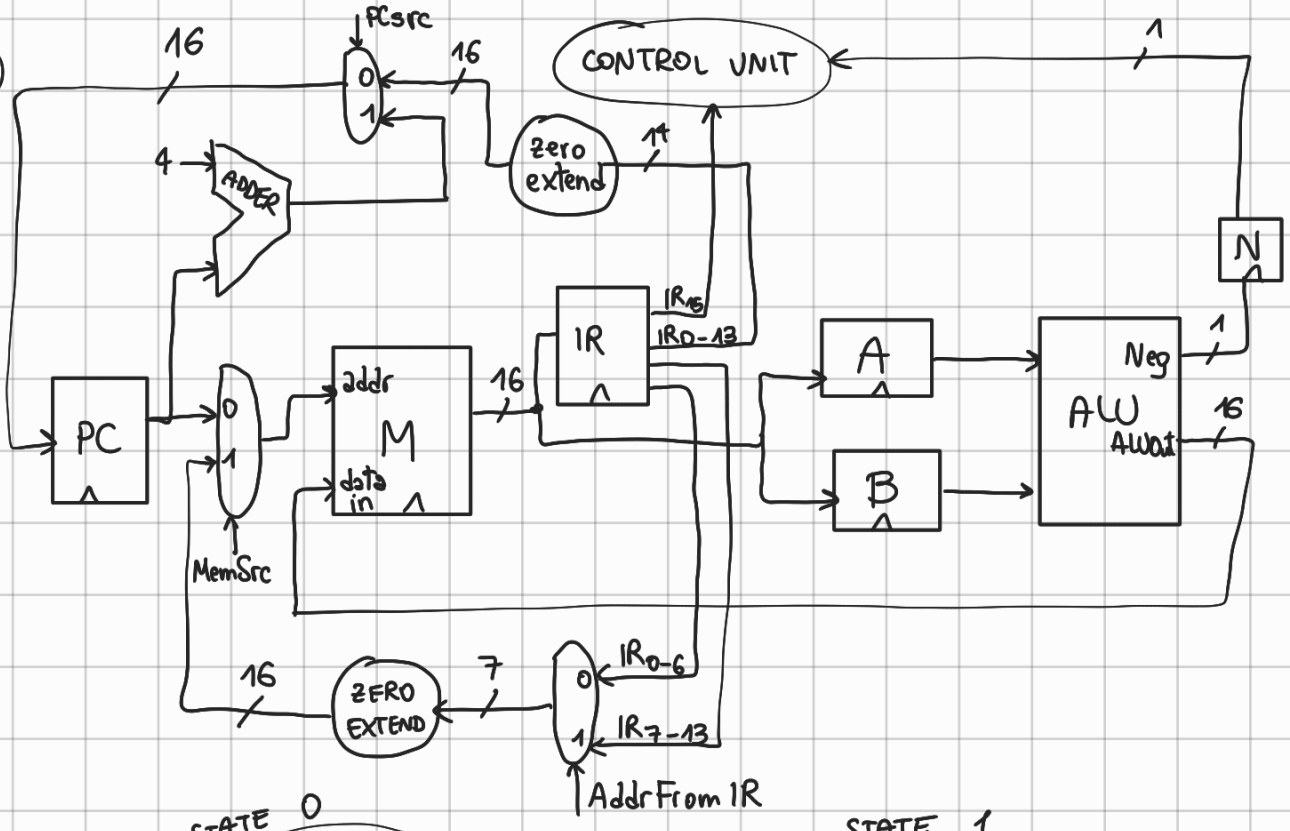
} Assuming Memory's output can be the input to A and B

- cycle 4: $ALUOut = A - B$
- cycle 5: $Mem[zero\text{-}ext(IR_{0-6})] = ALUOut$

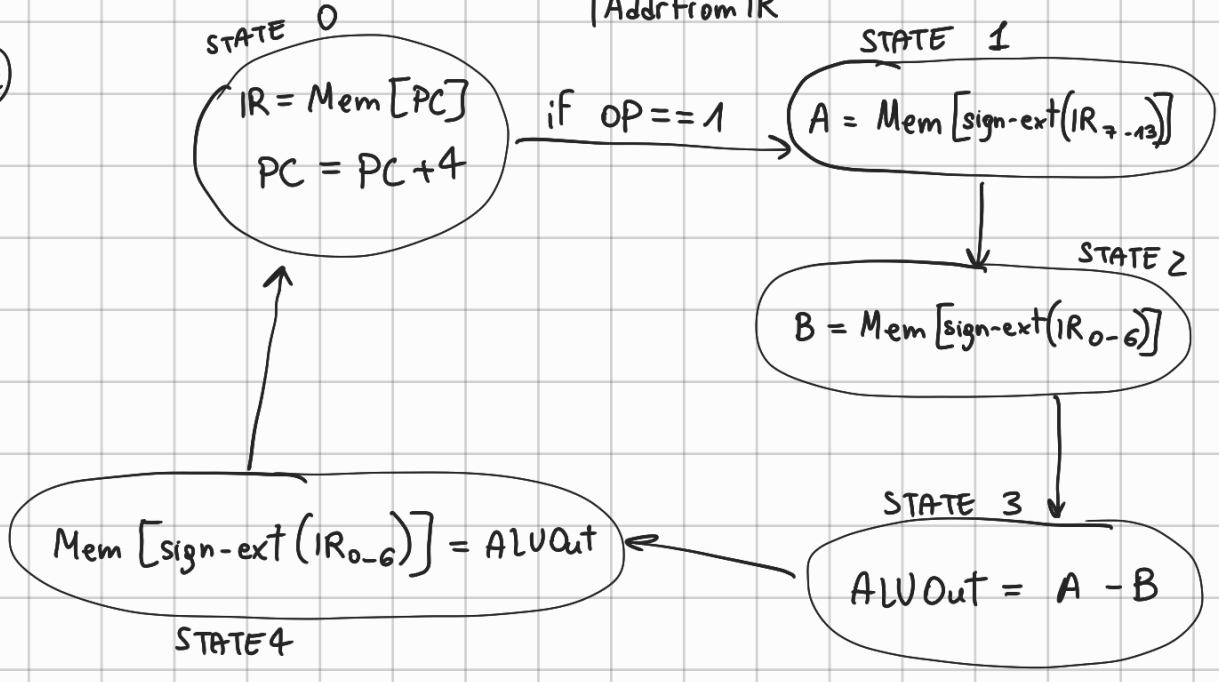
JIN instruction:

- cycle 1: $IR = Mem[PC]$, $PC = PC + 4$
- cycle 2: if $N == 1$: $PC = zero\text{-}ext(IR_{0-13})$

(1b)



(1c)



STATE 1: MemRead = 1, AddrFromIR = 1, Awrite = 1

STATE 2: MemRead = 1, AddrFromIR = 0, Bwrite = 1

STATE 4: MemWrite = 1, AddrFromIR = 0

(2b) $128\ MB = 128 \cdot 2^{20}\ B = 2^7 \cdot 2^{20}\ B = 2^{27}\ B$

CACHE SIZE $= 64\ KB = 64 \cdot 2^{10}\ B = 2^6 \cdot 2^{10}\ B = 2^{16}\ B$

(i) $\#\ CACHE\ LINES = \frac{CACHE\ SIZE}{LINE\ SIZE} = \frac{2^{16}\ B}{2^4\ B} = 2^{12}$ lines

(ii) Since the cache is direct-mapped, there is exactly one line per set. Hence $\#\ CACHE\ LINES = \#\ SETS$.

A memory address for this cache is partitioned as:

$\#\ OFFSET\ BITS = \log_2(LINE\ SIZE) = 4$ bits

$\#\ SET\ INDEX = \log_2(\#\ SETS) = \log_2(\#\ CACHE\ LINES) = 12$ bits

$\#\ TAG\ BITS = 27 - 12 - 4 = 11$ bits

(Recall that the division is | TAG | SET INDEX | OFFSET |)

(2a) (i) We access array A at index %rdx. Hence rdx must have the form:

$\%rdx = i \cdot N + j$

From line 4 we know that:

$\%rdx = j + 4i$

Putting things together we have: $\boxed{N = 4}$

Same reasoning for B.

%rdi = index to which we access B $\Rightarrow$ %rdi matches the pattern:

$\%rdi = j \cdot Q + i$

Also:

$\%rdi = \%rdi + \%rax = \%rdi + 5 \cdot \%rsi = i + 5 \cdot j$

Hence: $\boxed{Q = 5}$

Since arrays are stored in consecutive chunks of memory, we can deduce M from the addresses in which the first elements of the arrays are stored (last 2 lines of the assembly code).

Assuming 48 and 80 are in decimal notation, A is 32 bytes in total. Therefore the number of elements in A is $\frac{32}{sizeof(int)} = 8$. Since $N = 4$ we must have $M = 2$ because $M \cdot N = 8$ elements.

However, we cannot determine P because we don't know how much memory has been allocated for array B.

(ii)

```
int half_increment (int x) {
    int result =  x;
    if ( x==0 ){
        return result;
    } else {
        result = div(&x , 2);
        return result +x;
    }
```