# State Machine

**bflag controls value going on to the main bus**

3'd0:B\_bus={8'b00000000, DRAM}; // update from DRAM

3'd1:B\_bus=PC; // update from PC

3'd2:B\_bus=R1; // update from General Purpose Registers

3'd3:B\_bus=R2;

3'd4:B\_bus=R3;

3'd5:B\_bus=R;

3'd6:B\_bus=AC; // update from ALU output

3'd7:B\_bus={8'b00000000, IRAM}; // update from internal RAM

**cflag controls the registers to write bus onto**

0: dram  
1: ac  
2: r register (used to support add / subtract operations in ALU)  
3: gp register 3  
4: gp register 2  
5: gp register 1  
6: program counter  
7: address register

# ISA

**Note on Registers**

All instructions are stored on IRAM (8 bit data and address bus), so only the first 8 bits of the PC is fed to it as addresses. However, PC is a 16 bit register. Data on IRAM is loaded to the 8 bit IR. (ISA needs to be changed to store instructions on the DRAM and access from there).

**Registers available**

1. AR: 16 bit
2. R: 16 bit
3. IR: 8 bit
4. PC, R1, R2, R3, AC: 16 bit (can increment in one clock cycle)

**Instructions**

1. Fetch: fetch cycle (not visible to outside)
2. CLAC: clears AC
3. STAC: stores AC to DRAM location in AR
4. LDAC: loads DRAM location in AR to AC
5. MVAC: moves AC value to a register
   1. R1, R2, R3
   2. R
   3. AR
6. INCAC: increases AC by 1
7. DECAC: decreases AC by 1
8. ADD: AC+R
9. SUB: AC-R
10. MUL4: AC\*4
11. DIV: AC/2
12. JUMPNZ
    1. Z==1: PC=PC+1
    2. Z==0: (instruction after JUMPNZ in IRAM loaded to PC)
13. ADDM: value right after in IRAM is added to AC (AC = AC + k)
14. INCR1, INCR2, INCR3: R1 = R1 + 1
15. NOP: Does nothing, move to fetch
16. END: Stops the clock and ends processing

**State Diagram Implementation**

# Fetch

First Time: init all to zero, PC=0, read, IR<=M, PC+=1, SM<=IR

Fetch contains four states (currently works on four clock cycles)

1. Load instruction to bus from IRAM
2. Write bus value to IR
3. Increase PC
4. Load IR to state machine

# CLAC

Makes output of ALU zero and stores that in accumulator (all in one clock cycle)

# STAC