MANAGEMENT AND ANALYSIS OF PHYSICS DATASET (MOD. A)

SAP-2 and SAP-3 computer

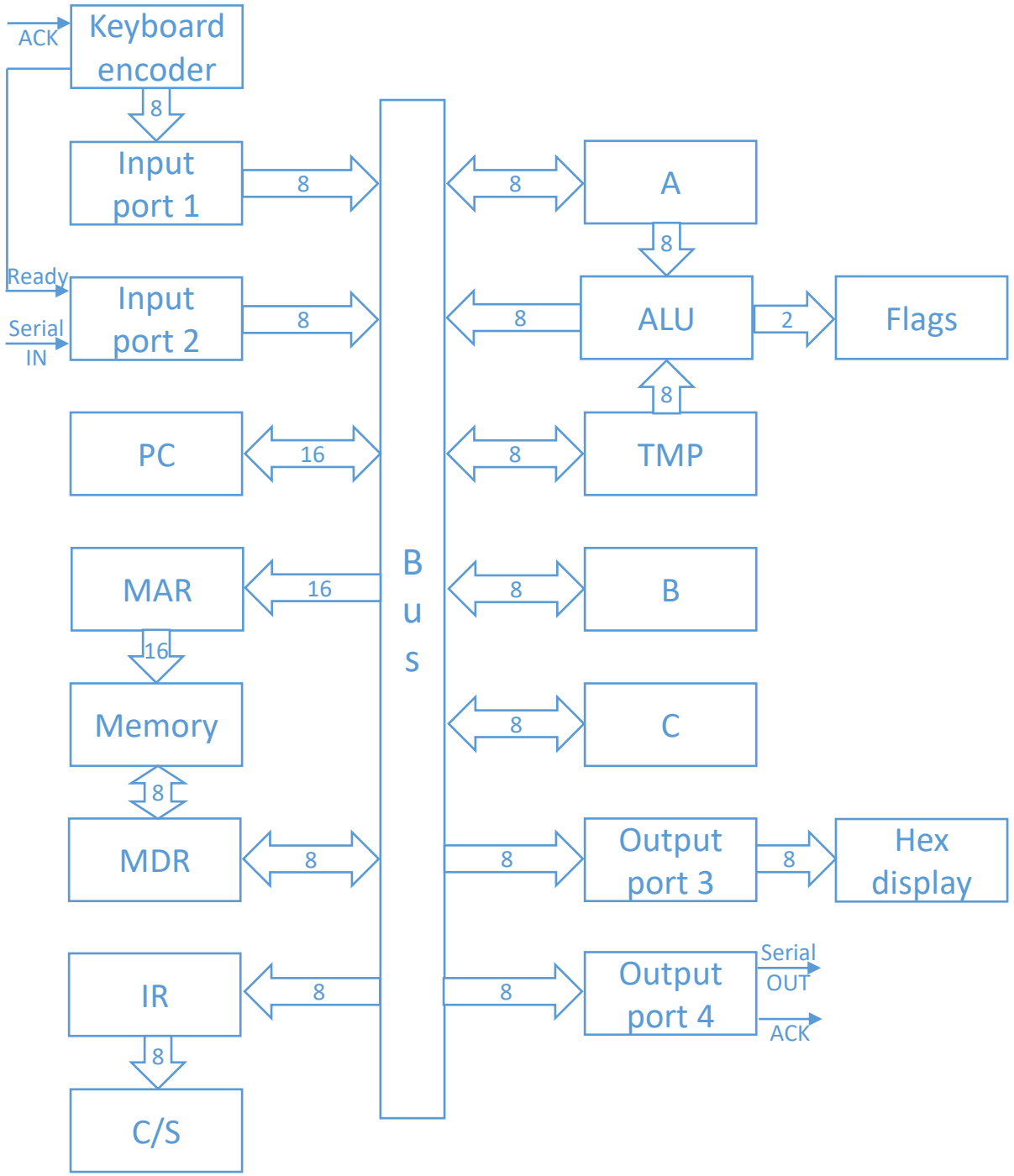
SAP-2

SAP-2

- SAP-1 is a computer
  - Store program and data before calculation
  - Automatically carry out program instructions
- SAP-2 is an evolution towards modern computer
  - Jump instructions give much more computing power

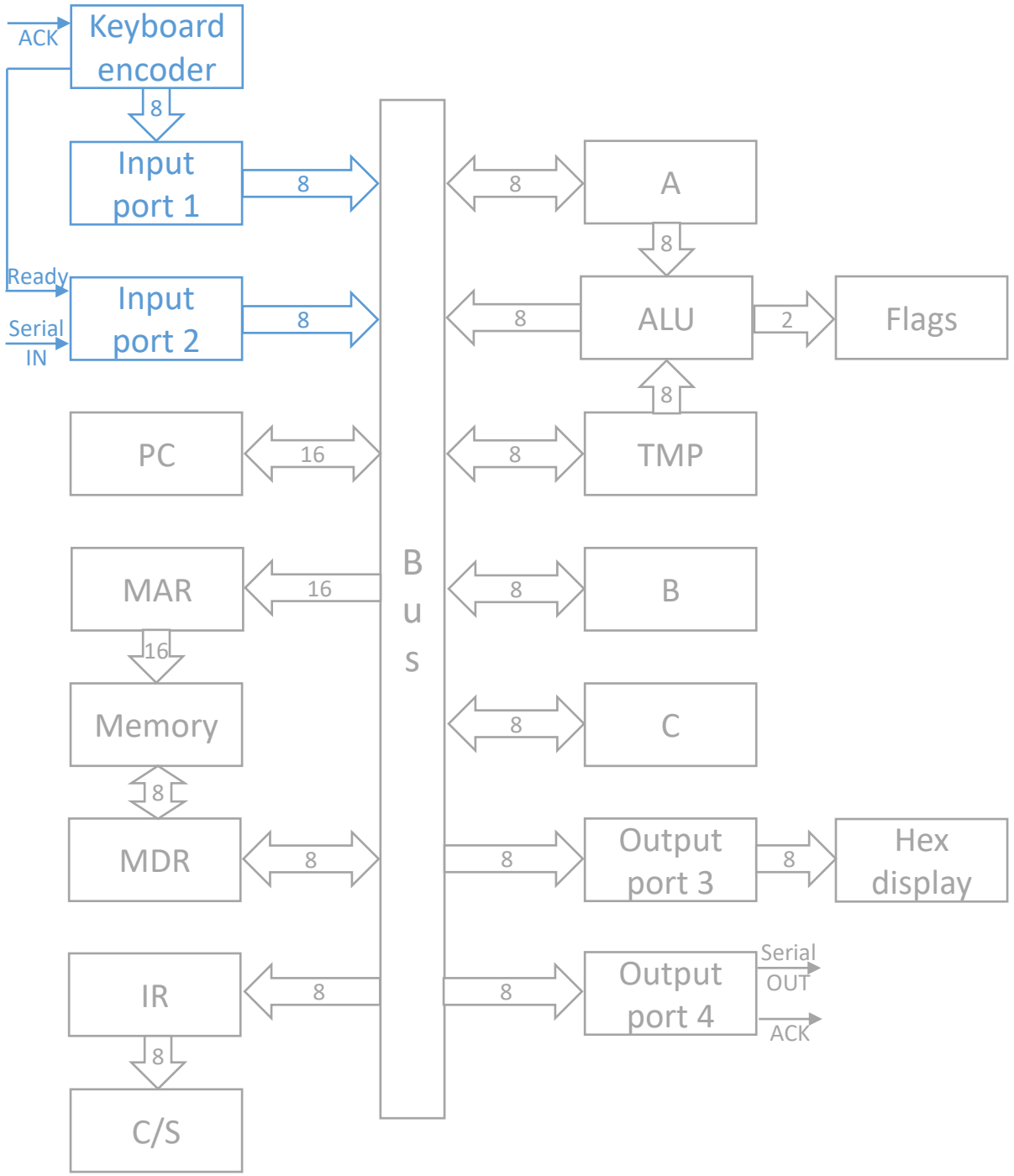
Architecture

- Simplified schema without clock, reset and controls
- All the register outputs to the bus are three-state
- Adopt bidirectional registers



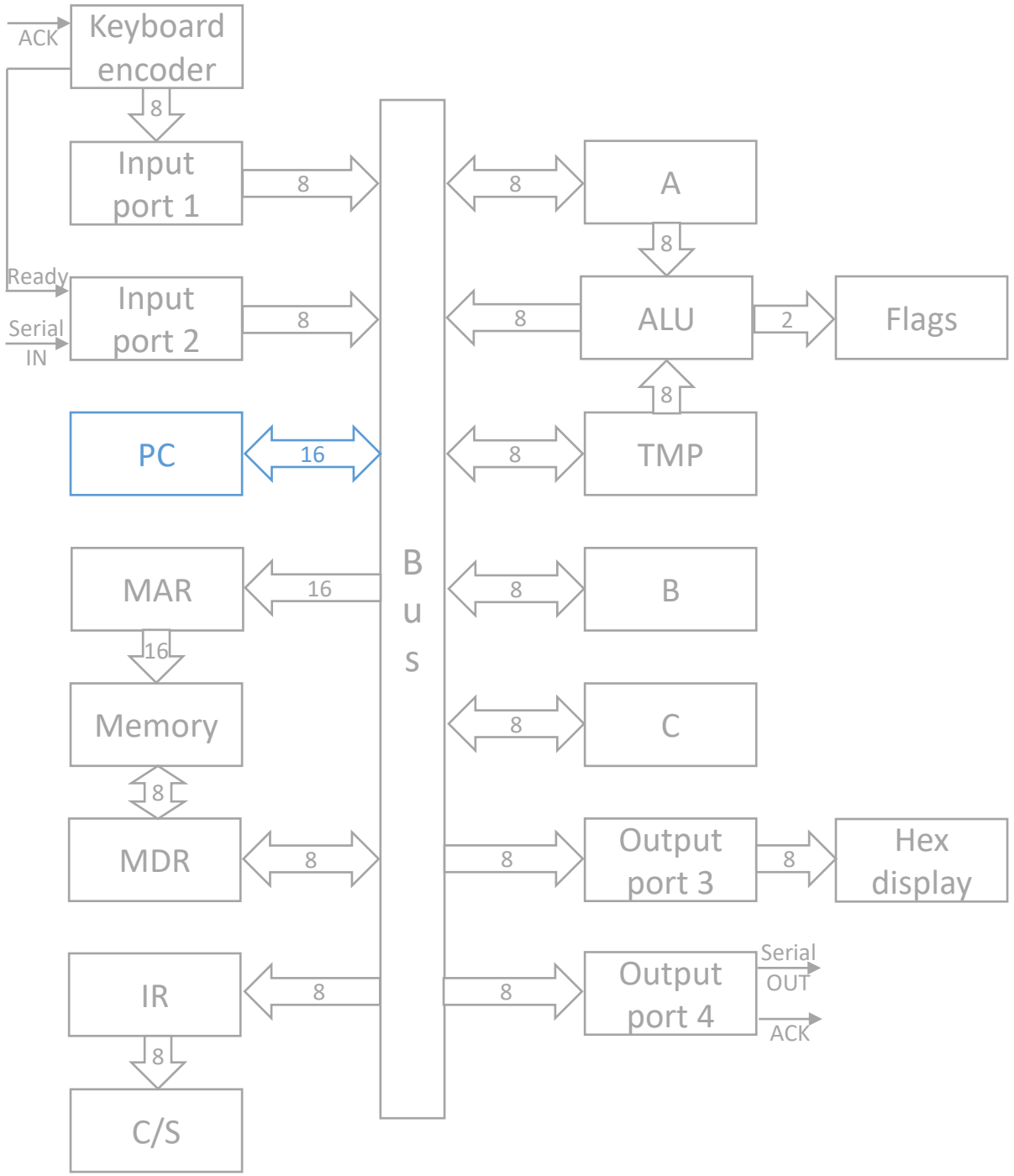
Architecture

- There are two input ports
- A hexadecimal keyboard encoder allows to enter hex instructions and data through port 1
- A serial input is connected to port 2



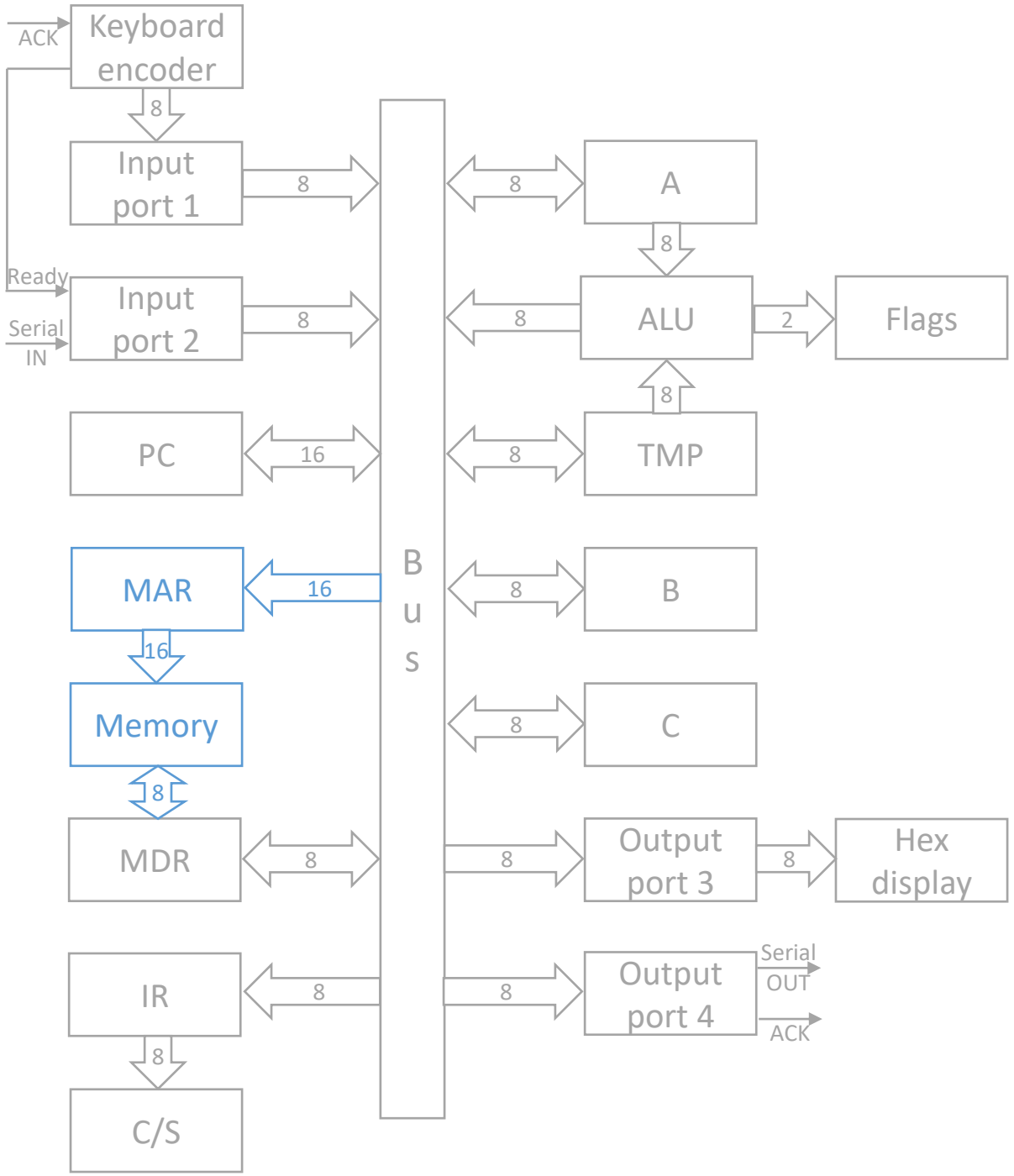
Architecture

- Program counter has the same role has for SAP-1 but it is 16-bit wide
- It counts from 0x0000 to 0xFFFF
- Before a run it is reset to 0x0000
- The connection to the bus is bidirectional



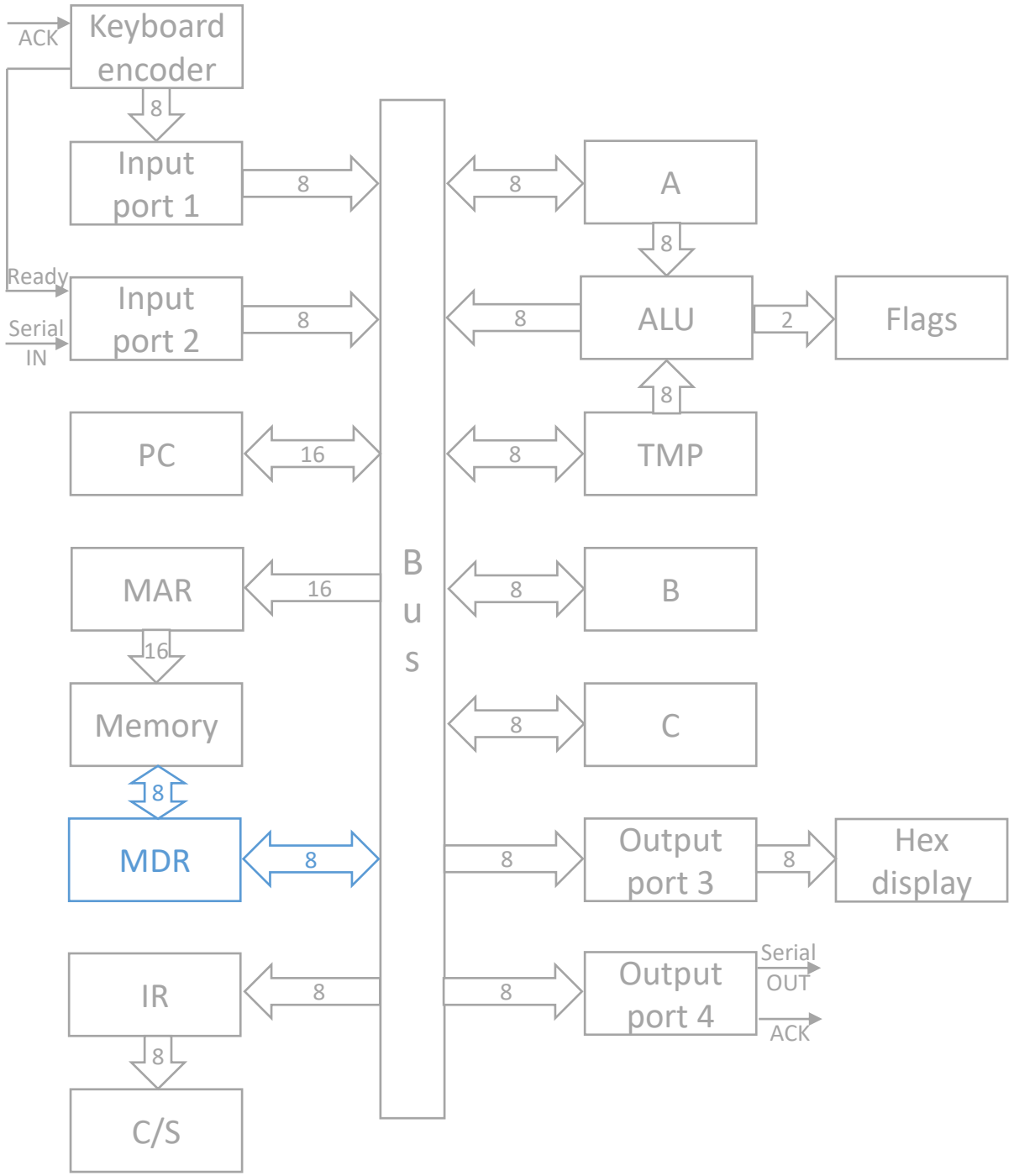
Architecture

- During fetch cycle MAR receive an address from PC and addresses a 64KB memory
- The memory has an initial 2KB ROM (0x0 to 0x7FF) followed by 62KB RAM (0x800 to 0xFFFF)
- ROM contains a program that initializes SAP-2 at power-up and interpret the keyboard inputs



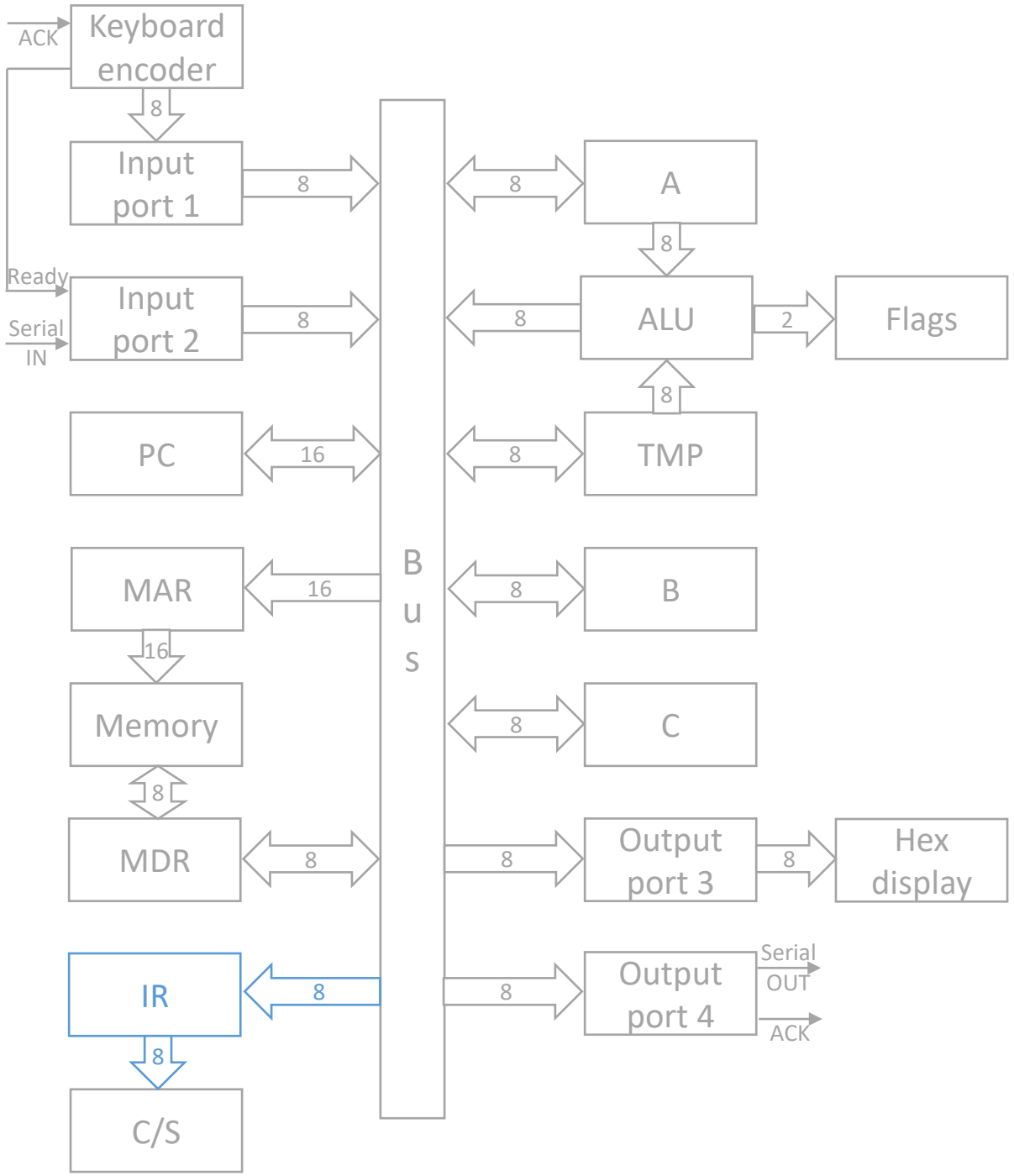
Architecture

- The Memory Data Register (MDR) is a 8-bit buffer register
- It receives data from the bus for writing to the RAM and it sends data to the bus for reading



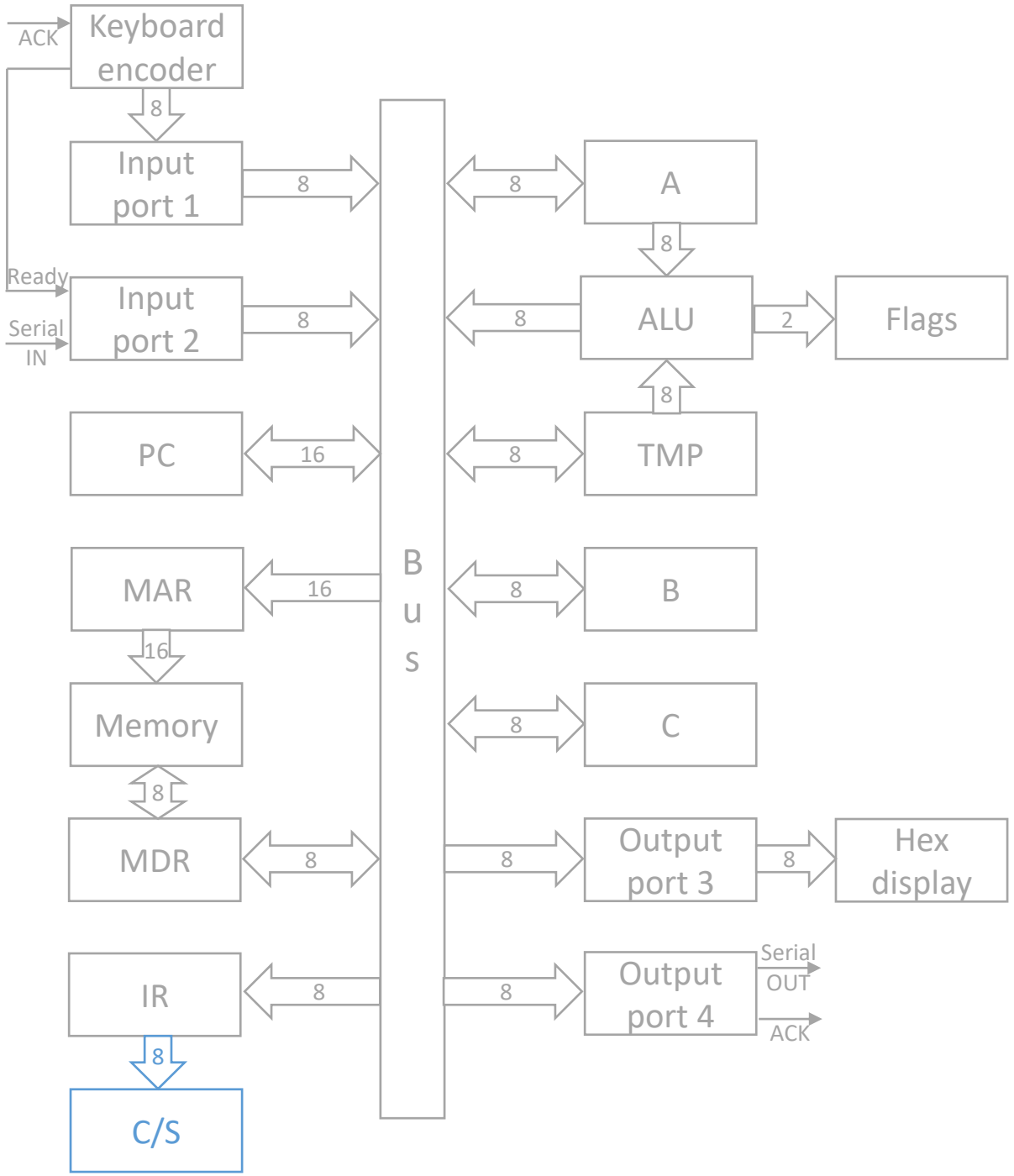
Architecture

- The instruction register use an 8-bit op code because the number of instructions is grater than in SAP-1



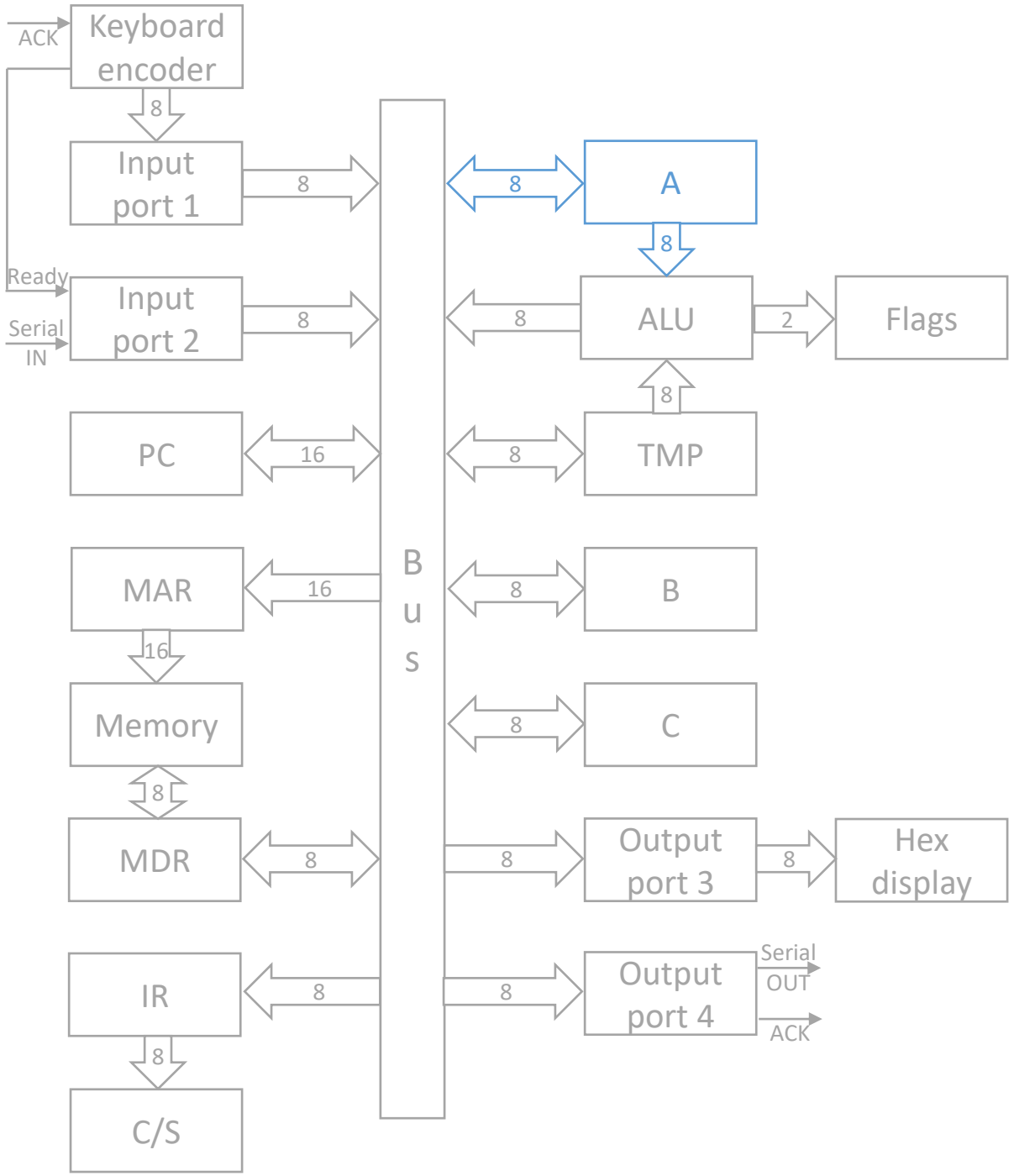
Architecture

- Except from the fact that the control word is bigger, it is conceptually identical to the one of SAP-1



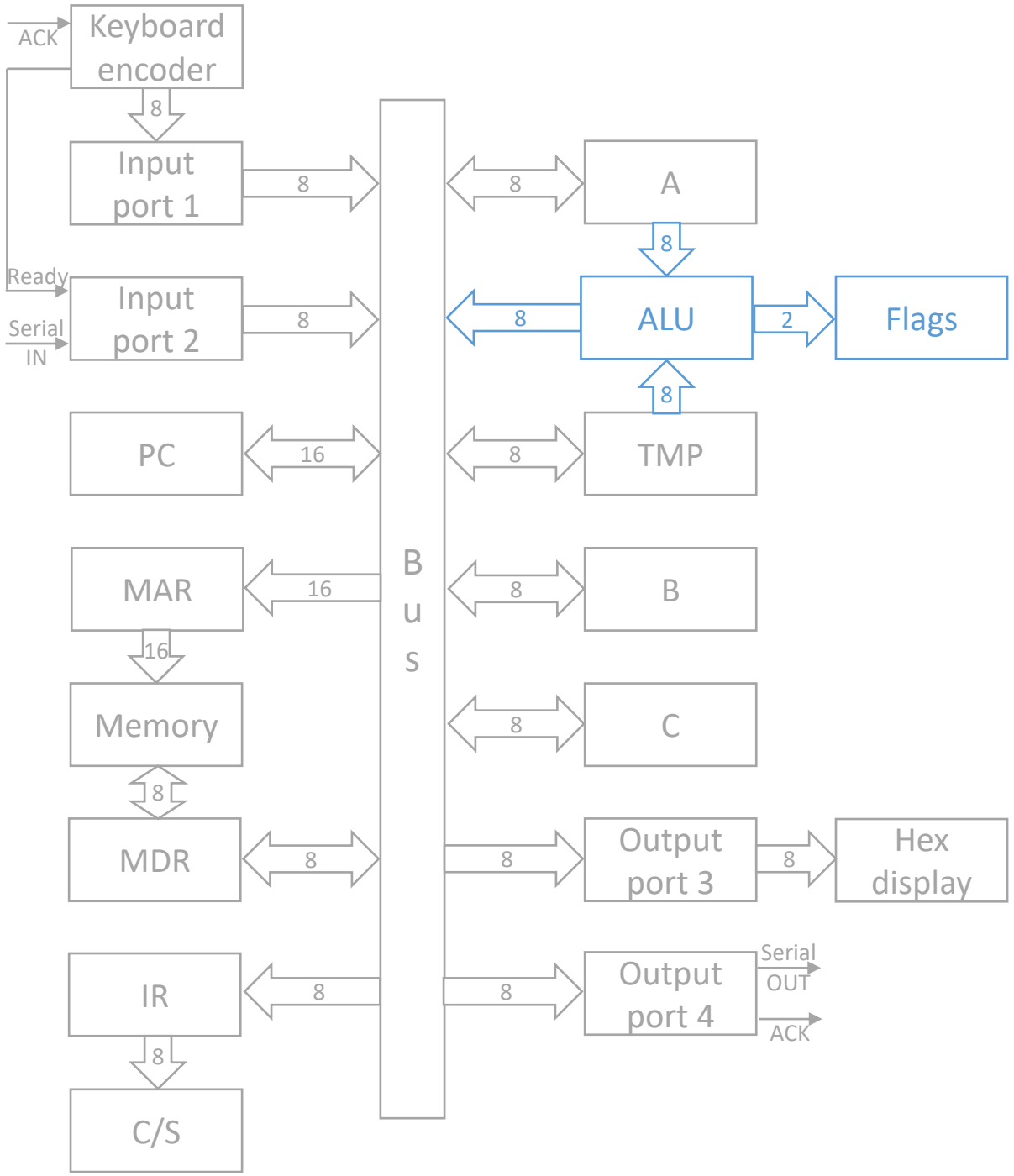
Architecture

- The accumulator is identical to the one of SAP-1



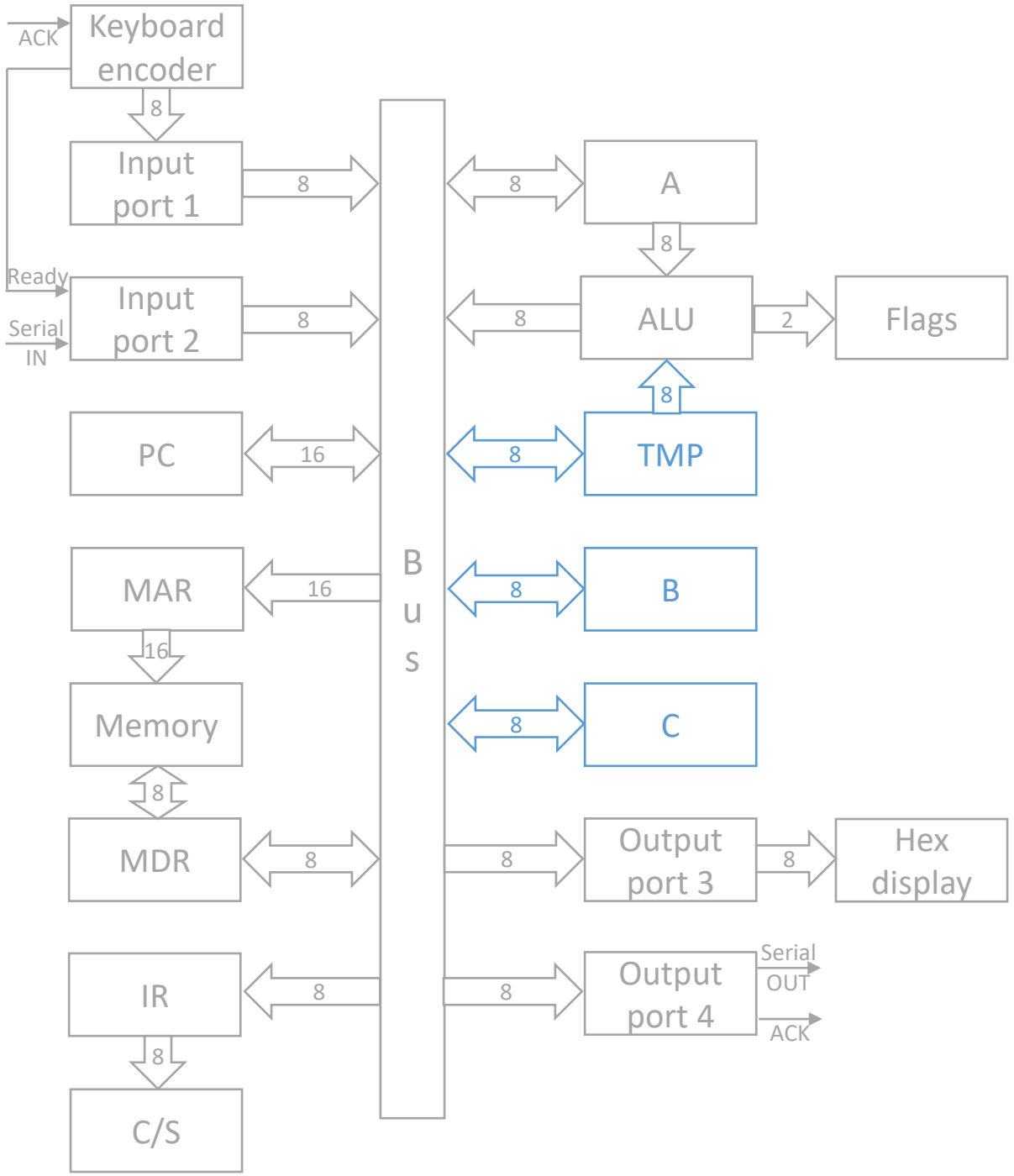
Architecture

- ALU can perform arithmetic and logic operations
- It has some control bits that determine the operation
- Flag is a flip-flop that can keep track of a changing condition
- A Sign flag is set when the accumulator become negative
- A Zero flag is set when the accumulator become zero
- All the instructions that use the ALU can affect the flags



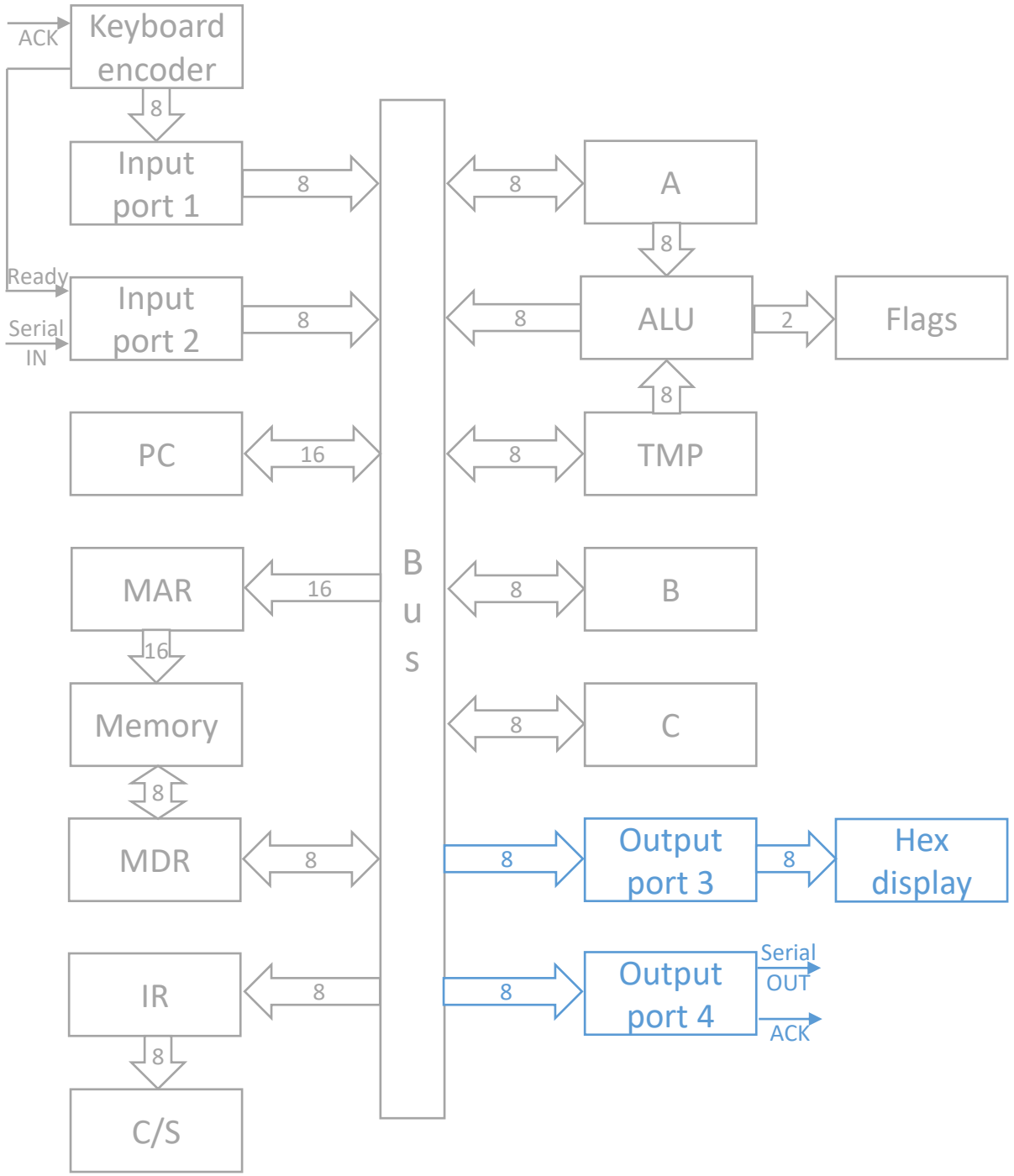
Architecture

- A temporary register TMP provide data to the ALU
- It gives more freedom in the calculation
- There is one more register (C) in SAP-2



Architecture

- There are two output ports
- A hexadecimal display is driven by port 3
- A serial link is fed by port 4
- Acknowledge (ACK) and ready are two signals used for handshaking



Timing states

- The fetch cycle is the same as the one of SAP-1
- The timing states of the execution cycle are different for each instruction and can involve the use of memory (memory-reference instruction)
- The instruction set is built by 43 instructions

LDA and STA

- Load the accumulator is the same as before but the operand is a 16-bit address
- STA means STore the Accumulator
- It needs three bytes to be stored in memory: the op code and the 16-bit operand
- Three memory addresses are used to store LDA or STA instruction
- For instance, “STA 0x00A8” means to copy the content of the accumulator to memory address 0x00A8

$$A = 00010011 \xrightarrow{\text{STA 0x00A8}} M_{168} = 00010011$$

MVI X

- MVI means MoVe Immediate
- It loads a designed register X={A,B,C} with some data
- It needs two bytes to be stored in memory: the op code and the 8-bit operand
- Two memory addresses are used to store a MVI X instruction
- For instance, “MVI B 0x35” means to load the content of register B with the byte 0x35

$$\xrightarrow{\text{MVI B 0x35}} B = 00110101$$

MOV X Y

- Instructions that move the data without passing through memory are called register instructions
- MOV stands for MOVe
- It copies the data from one register Y={A,B,C} to another register X={A,B,C} with X≠Y
- For instance, “MOV A B” means to load the content of register B to register A

$$A = 0x35 \quad B = 0x81 \xrightarrow{\text{MOV A B}} A = 0x81 \quad B = 0x81$$

ADD X and SUB X

- It adds/subtracts the content of register X={A,B,C} to/from the accumulator content
- For instance, “ADD B” means to sum the content of register B to the accumulator register

$$A = 0x35 \quad B = 0x81 \xrightarrow{\text{ADD B}} A = 0xB6 \quad B = 0x81$$

INR X and DCR X

- INR stands for INcRement and DCR for DeCRement
- It increments/decrements by one the content of register X={A,B,C}
- For instance, “INR C” means to add one to the content of register C

$$C = 0x1D \xrightarrow{\text{INR C}} C = 0x1E$$

- These instructions use the accumulator for computing the increment/decrement and then they send back the result to the corresponding register

JMP

- Instructions that change the program sequence are called jump instructions
- JMP stands for JuMP
- It tells to get the instruction from a particular memory address
- It needs a 2-byte operand
- For instance, “JMP 0x0020” at address 0x0005 means to execute as next instruction the one stored at memory address 0x0020

$$PC = 0x0006 \xrightarrow{\text{JMP 0x0020}} PC = 0x0020$$

JM

- JM means Jump if Minus
- The jump is done if and only if the sign flag is set (conditional jump)
- Sign flag is defined as follow

$$S = \begin{cases} 0 & if\ A \geq 0 \\ 1 & if\ A < 0 \end{cases}$$

- For instance, “JM 0x0020” at address 0x0005 means to execute as next instruction the one stored at memory address 0x0020 if S=1, otherwise the next instruction to be executed is 0x0006

$$PC = 0x0006 \xrightarrow{\text{JM 0x0020}} \begin{cases} PC = 0x0020 & if\ S = 1 \\ PC = 0x0006 & if\ S = 0 \end{cases}$$

JZ

- JZ means Jump if Zero
- The jump is done if and only if the zero flag is set (conditional jump)
- Zero flag is defined as follow

$$Z = \begin{cases} 0 & if\ A \neq 0 \\ 1 & if\ A = 0 \end{cases}$$

- For instance, “JZ 0x0020” at address 0x0005 means to execute as next instruction the one stored at memory address 0x0020 if Z=1, otherwise the next instruction to be executed is 0x0006

$$PC = 0x0006 \xrightarrow{\text{JZ 0x0020}} \begin{cases} PC = 0x0020 & if\ Z = 1 \\ PC = 0x0006 & if\ Z = 0 \end{cases}$$

JNZ

- JNZ means Jump if Not Zero
- The jump is done if and only if the zero flag is not set (conditional jump)
- Zero flag is defined as follow

$$Z = \begin{cases} 0 & if\ A \neq 0 \\ 1 & if\ A = 0 \end{cases}$$

- For instance, “JNZ 0x0020” at address 0x0005 means to execute as next instruction the one stored at memory address 0x0020 if Z=0, otherwise the next instruction to be executed is 0x0006

$$PC = 0x0006 \xrightarrow{\text{JNZ 0x0020}} \begin{cases} PC = 0x0020 & if\ Z = 0 \\ PC = 0x0006 & if\ Z = 1 \end{cases}$$

CALL and RET

- CALL stands for CALL the subroutine
- RET stands for RETurn
- A subroutine is a program stored in memory for possible use in another program
- CALL must include the starting address of the desired subroutine
- RET is used at the end of the subroutine to go back to the original program
- When CALL is executed the content of the PC is stored in the last two memory addresses. The operand of CALL is then loaded in the PC
- When RET is executed, the address loaded in the last two memory addresses is loaded back into the PC

Logic instructions

- Besides arithmetic, SAP-2 can do logic
- CMA (CoMplement the Accumulator)
  - inverts each bit of the accumulator
- ANA X (ANd the Accumulator)
  - bitwise AND between the accumulator and register X={B,C}
- ORA X (OR the Accumulator)
  - bitwise OR between the accumulator and register X={B,C}
- XRA X (XoR the Accumulator)
  - bitwise XOR between the accumulator and register X={B,C}

Logic instructions

- Each operator has also an immediate version of the instruction
- ANI (ANd Immediate)
  - bitwise AND between the accumulator and one byte given as operand
- ORI (OR Immediate)
  - bitwise OR between the accumulator and one byte given as operand
- XRI (XoR Immediate)
  - bitwise XOR between the accumulator and one byte given as operand

Miscellaneous instructions

- NOP (No Operation)
  - It is used for delaying data processing
  - Four timing states are needed to fetch and execute a NOP
- HLT (HaLT)
  - As for SAP-1
- IN (INput) and OUT (OUTput)
  - It transfers the data from/to a specific port to/from the accumulator
  - The port is specified in the operand
- RAL (Rotate the Accumulator Left) and RAR (Rotate the Accumulator Right)
  - Shift all accumulator bit to the left/right and move the MSB/LSB to the LSB/MSB

$$A = 0x35 \xrightarrow{\text{RAL}} A = 0x6A$$

Op Code

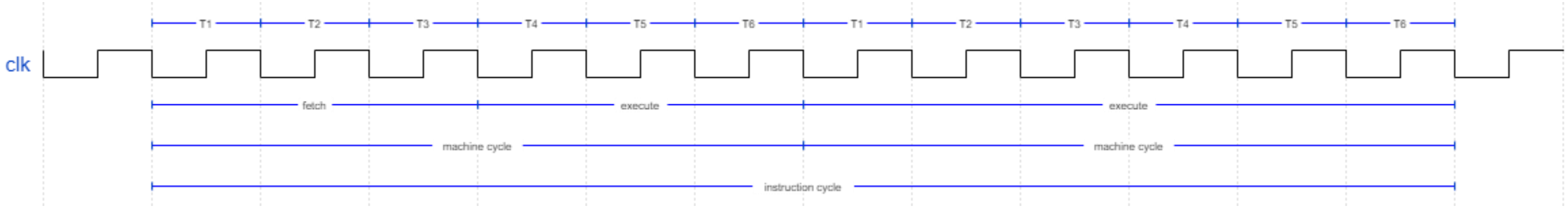
Mnemonics	Op Code
ADD B	0x80
ADD C	0x81
ANA B	0xA0
ANA C	0xA1
ANI byte	0xE6
CALL address	0xCD
CMA	0x2F
DCR A	0x3D
DCR B	0x05
DCR C	0x0D
HLT	0x76
IN byte	0xDB
INR A	0x3C
INR B	0x04
INR C	0x0C

Mnemonics	Op Code
JM address	0xFA
JMP address	0xC3
JMZ address	0xC2
JZ address	0xCA
LDA address	0x3A
MOV A,B	0x78
MOV A,C	0x79
MOV B,A	0x47
MOV B,C	0x41
MOV C,A	0x4F
MOV C,B	0x48
MVI A byte	0x3E
MVI B byte	0x06
MVI C byte	0x0E
NOP	0x00

Mnemonics	Op Code
ORA B	0xB0
ORA C	0xB1
ORI byte	0xF6
OUT byte	0xD3
RAL	0x17
RAR	0x1F
RET	0xC9
STA address	0x32
SUB B	0x90
SUB C	0x91
XRA B	0xA8
XRA C	0xA9
XRI byte	0xEE

Machine/Instruction cycle

- In SAP-2 some instructions takes more than one machine cycle to fetch and execute
- CALL is the more complex instruction that require 18 timing states that correspond to 3 machine cycle for one instruction cycle
- SAP-2 C/S has a variable machine cycle



Exercises

- How many times the DCR instruction is executed in the following program? How much space is needed to store it?

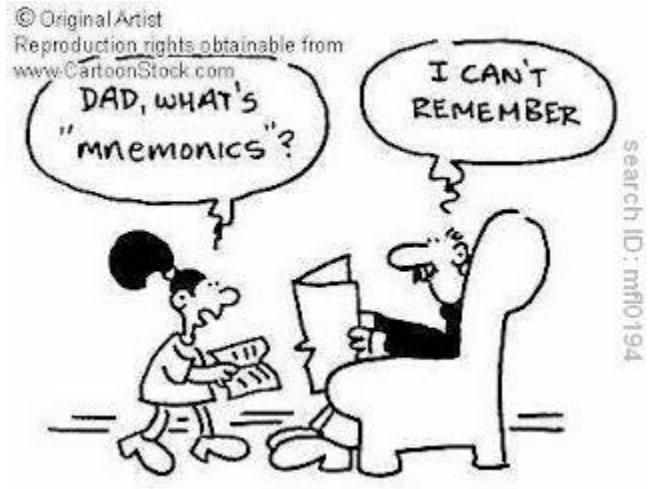
```
MVI C 0x03
DCR C
JZ 0x0009
JMP 0x0002
HLT
```

- Write a program that multiplies the two numbers 11_{10} and 5_{10}
- Write a program that read from input port 2 and if the LSB is a 1 it loads the accumulator with ASCII character *Y* otherwise with *N.* Finally, it sends the results to output port 3

SAP-3

SAP-3

- An increased instruction set
- Additional registers (D, E, H and L) equivalent to SAP-2 registers
- F register for storing flags value
- New flags
- Stack pointer: a 16-bit register that controls a dedicated portion of memory



Old instructions

- MOV and MVI
  - Are the same but with more registers
  - More registers means to be faster because register instructions use less timing states than memory reference instructions
- INC and DCR
- ANA, ORA and XRA
- ANI, ORI and XRI
- JMP, JM, JZ and JNZ

New flags

- Carry (CY) flag
  - In order to detect overflow of the accumulator a carry flag is used
  - In output of the 8-bit adder/subtracter there is a carry bit that in the previous versions was disregarded
  - SAP-3 uses this bit called carry for sum and borrow for subtraction as a flag
  - It can be considered as the ninth bit of the accumulator
- Parity (P) flag
  - P is set if the number of 1s of the result of the ALU is even

New instructions

- STC (SeT Carry)
  - It sets the carry flag if it is not already set
- CMC (CoMplement the Carry)
  - It sets the carry to its complement
  - STC followed by CMC resets the carry
- ADD X and SUB X are the same but setting the carry flag
- ADC X (Add with Carry) and SBB (SuBtract with Borrow)
  - It adds/subtracts the content of a register to/from the accumulator plus/minus the carry flag

$$A = 0x83 \quad E = 0x12 \quad CY = 1 \xrightarrow{\text{ADC E}} A = 0x96 \quad E = 0x12 \quad CY = 0$$

$$A = 0xFF \quad E = 0x02 \quad CY = 1 \xrightarrow{\text{SBB E}} A = 0xFC \quad E = 0x02 \quad CY = 0$$

New instructions

- RAL (Rotate All Left) and RAR (Rotate All Right)
  - Rotate left/right all the bit of the accumulator including the carry flag

$$CY = 1 \quad A = 0x74 \xrightarrow{\text{RAL}} CY = 0 \quad A = 0xE9$$

- RLC (Rotate Left with Carry) and RRC (Rotate Right with Carry)
  - Rotated left/right the accumulator bit and the MSB/LSB is saved in the carry flag

$$CY = 1 \quad A = 0x74 \xrightarrow{\text{RLC}} CY = 0 \quad A = 0xE8$$

New instructions

- CMP X (CoMPare)
  - Compare the content of a register with the content of the accumulator
  - The outcome can be seen from the zero flag

$$Z = \begin{cases} 0 & if\ A \neq X \\ 1 & if\ A = X \end{cases}$$

- ADI, ACI, SUI, SBI and CPI are immediate instructions
  - They need a byte as operand
  - ADI/ACI adds the operand to the accumulator without/with the carry flag
  - SUI/SBI subtracts the operand and the carry flag from the accumulator
  - CPI compares the immediate byte with the accumulator

New instructions

- JP (Jump if Positive)
  - It has the opposite effect of JM
  - It jumps also if zero
- JC and JNC (Jump if Carry and Jump if Not Carry)
  - JC (JNC) jumps if the carry flag is (is not) set
- JPE and JPO (Jump if Parity Even and Jump if Parity Odd)
  - JPE/JPO jumps in case P is set/reset

Extended register instructions

- Some instructions use pairs of registers in order to process 16-bit word
- Pairs are B-C, D-E and H-L so they are abbreviated B, D and H in the context of extended register
- LXI Y double-byte (Load the eXtended Immediate)
  - Load the operand to a specific extended register Y={B,D,H}
- DAD Y (Double Add)
  - Add the content of a specific extended register Y={B,D,H} to H
- INX Y and DCX Y (Increment and DeCrement the eXtended)
  - Increment/decrement a specific extended register Y={B,D,H}

Indirect instructions

- The extended register H is used as a data pointer
- Instead of using a direct addressing (like LDA X where X is a double-byte), they use a pointer to a memory location
- MOV X M
  - Load a specified register X={A,B,C,D,E,H,L} with the data addressed by HL

$$HL = 0x3404 \quad M_{0x3404} = 0x74 \xrightarrow{\text{MOV B M}} B = 0x74$$

- MOV M X
  - Load the memory location addressed by HL with the content of X={A,B,C,D,E,H,L}

$$HL = 0x3404 \quad C = 0x98 \xrightarrow{\text{MOV M C}} M_{0x3404} = 0x98$$

Indirect instructions

- MVI M byte
  - Load the memory location addressed by HL with a data byte

$$HL = 0x14A3 \xrightarrow{\text{MVI M 0xB1}} M_{0x14A3} = 0xB1$$

- All the following instructions are analogue to their register version, but they use instead the memory location at address HL
  - ADD M, ADC M, SUB M, SBB M, INR M, DCR M, ANA M, ORA M, XRA M and CMP M

Stack instructions

- Stack is a portion of memory used for storing the return address of subroutines
- In SAP-2 there is a small stack at the end of the memory: two memory addresses are used for storing the return address
- In SAP-3
  - The programmer can choose where to locate the stack and how big it is
  - There are special instructions for read and write into the stack
    - Stack instructions are indirect because they use a Stack Pointer (SP) similarly to HL
    - To initialize SP the immediate load instruction can be used: LXI SP double-byte

Stack instructions

- The PUSH Y instruction is used for saving the program status before calling a subroutine
  - The extended register is Y={B,D,H,PSW} where PSW is the program status word that is the concatenation of the registers A (Accumulator) and F (Flags)
  - When PUSH is executed
    - The stack pointer is decremented: SP-1
    - The high byte of Y is stored in M_{SP-1}
    - The stack pointer is decremented again: SP-2
    - The low byte of Y is stored in M_{SP-2}

Stack instructions

- The POP Y instruction is used for getting back the program status after a subroutine is called back
  - The extended register is Y={B,D,H,PSW} where PSW is the program status word that is the concatenation of the registers A (Accumulator) and F (Flags)
  - When POP is executed
    - The low byte of Y is loaded with the content of M_{SP}
    - The stack pointer is incremented: SP+1
    - The high byte of Y is loaded with the content of M_{SP+1}
    - The stack pointer is incremented again: SP+2

Stack instructions

- CALL is used for saving automatically return addresses
  - When executed the PC is pushed onto the stack and the starting address of the routine is loaded into the PC
- RET is used at the end of a subroutine for coming back to the previous address
  - When executed it pops the return address off the stack into the PC
- For instance:

```
0x2000    LXI SP 0x2100
0x2003    CALL 0x8050
0x2006    MVI A 0x0E
…
0x20EF    HLT
…
0x8050    …
…
0x8059    RET
```

- LXI and CALL takes 3 bytes because of the operand
- LXI load the SP with 0x2100
- CALL makes the address 0x2006 saved in the stack
  - SP is decremented
  - 0x20 is stored in 0x20FF
  - SP is decremented
  - 0x06 is stored in 0x20FE

- CALL makes the address 0x8050 loaded in PC
- At the end of the routine RET is executed
  - 0x06 is loaded into the low byte of PC
  - SP is incremented
  - 0x20 is loaded into the high byte of PC
  - SP is incremented

Stack instructions

- Call and return instructions can be conditional
- CNZ, CZ, CNC, CC, CPO, CPE, CP and CM
- RNZ, RZ, RNC, RC, RPO, RPE, RP and RM
- They are similar to conditional jumps with the same meanings (instead of J, C for call and R for return)

Exercises

- Write a program that adds 1200_{10} to 3400_{10} and store the result in registers H and L, with and without the use of extended register instructions
- Using pointers, write a program that moves all the data stored between memory address 0x2030 and 0x212F to the memory space between address 0x5030 and 0x512F