# Design of a Microprocessor

## The uP3207 processor

# Operation of a Simple Microprocessor

**Memory**

…

…

…

…

**Program image**

…

…

…

**Data / variables**

…

…

…

**Instruction Fetch Unit**

**Instruction decode and sequencing unit**

**ALU**

**Register File**

**I/O devices**

**CPU**
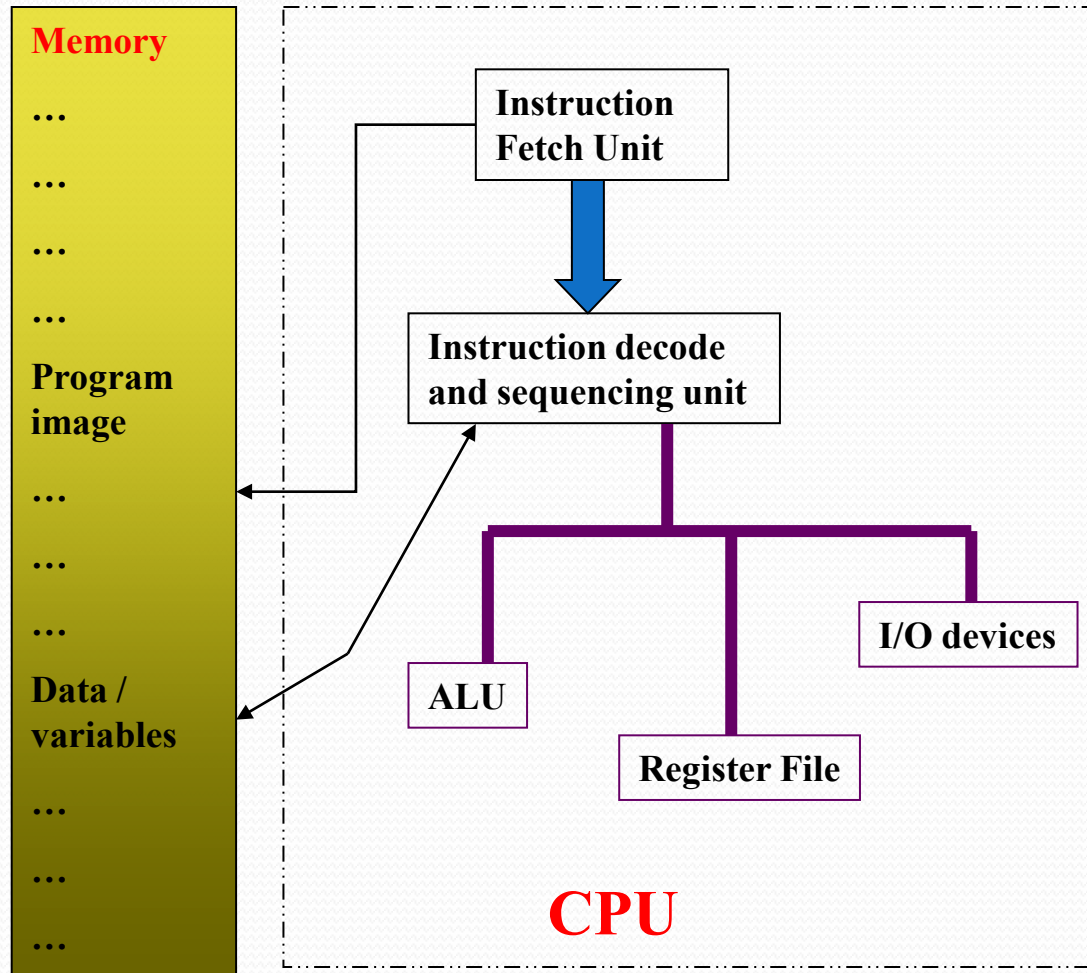
- Program to be executed is stored in memory
- Instruction Fetch Unit of CPU will fetch an instruction at a time from memory
- Instruction is sent to the decoding and sequencing unit (DSU)
- The DSC is a state machine that creates the sequence of actions to execute the instruction
  - The sequence acts on objects such the ALU, registers and I/O devices
- Once done control is reverted to the Instruction Fetch Unit to fetch the next instruction.
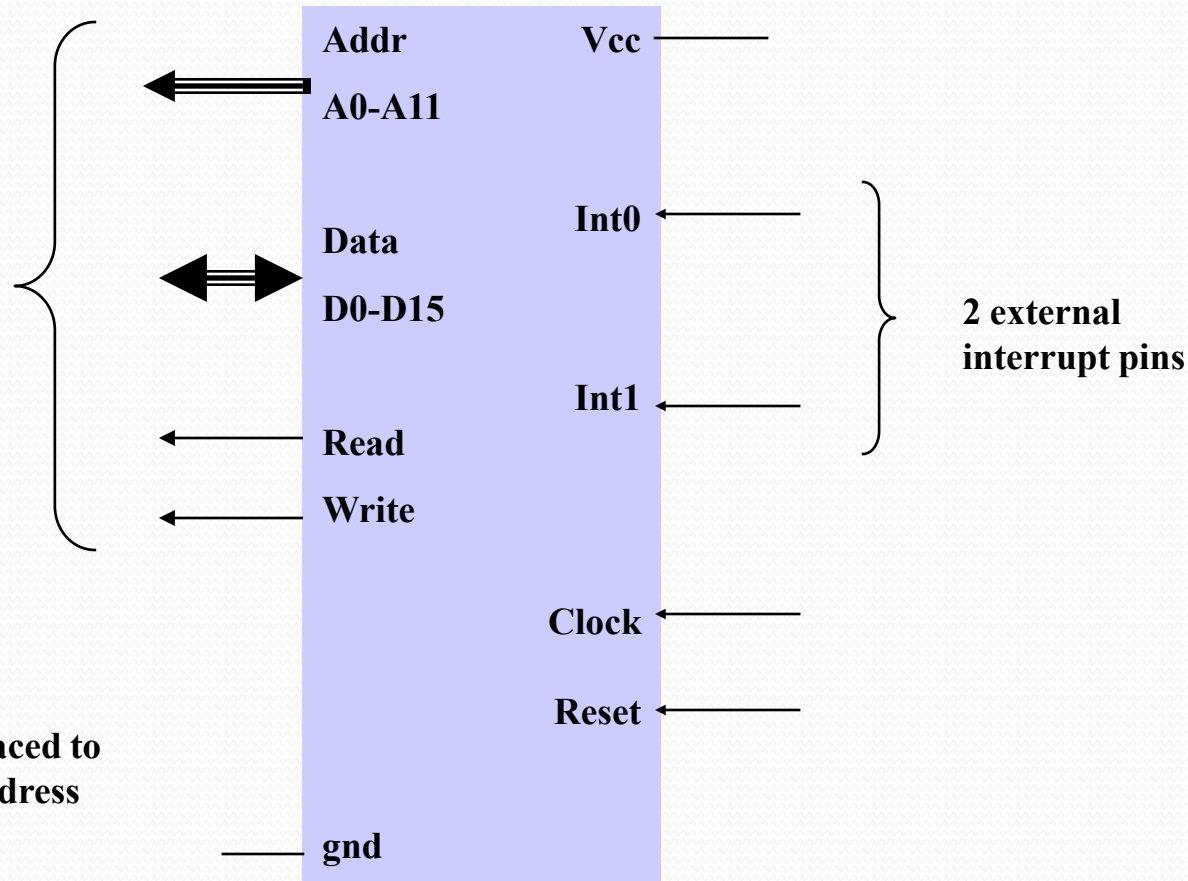
# The uP3207 processor

Our First Central Processing Unit

- ## What to design?
  - Requirement analysis on intended application
  - Not to be addressed in this course. Assume this is done.
- ## Features
  - 16 bit data width processor
  - 4K words addressable memory
  - Stack for temporary variable storage
  - Single word instructions
    - Single operands architecture
  - Support direct and indirect addressing
  - Support multiple interrupts from input and output devices
  - Low cost
  - Complete functionalities

# uP3207 Microprocessor

**Memory interface
4K words of 16 bit
memory, with separate
read/write control**

**Addr**

**A0-A11**

**Data**

**D0-D15**

**Read**

**Write**

**Vcc**

**Int0**

**Int1**

**Clock**

**Reset**

**gnd**

**2 external
interrupt pins**

**I/O devices will be interfaced to
the uP on the memory address
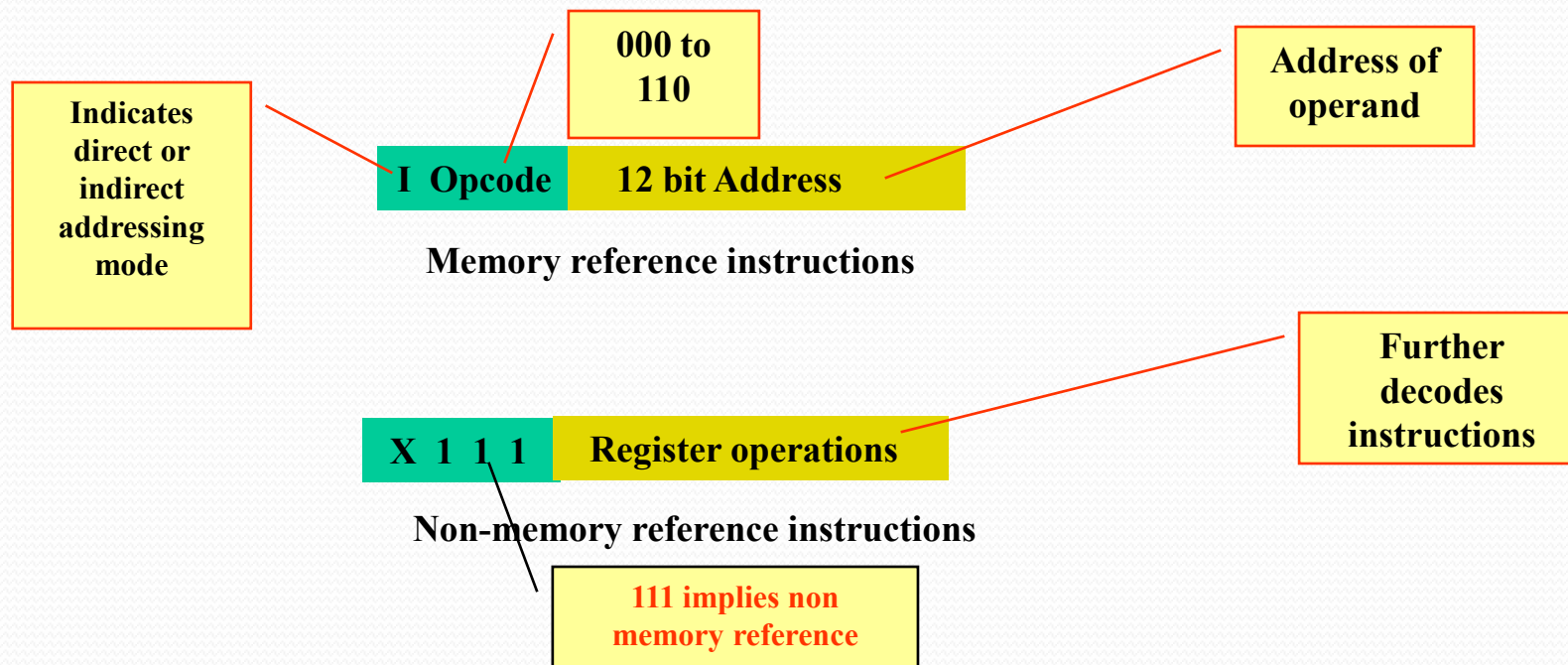space.**

# ISA: Registers and memory overview

- PC: program counter
  - Holds address of the next instruction
- SP: stack pointer
- BP: base pointer
- AC: accumulator
  - main processor register
- XR: operand register
  - For holding the operand of arithmetic/logic operations
- Flags
  - S=AC negative
  - Z=AC zero
  - C=carry flag set
  - Y=DR=0

**Programmer's View**
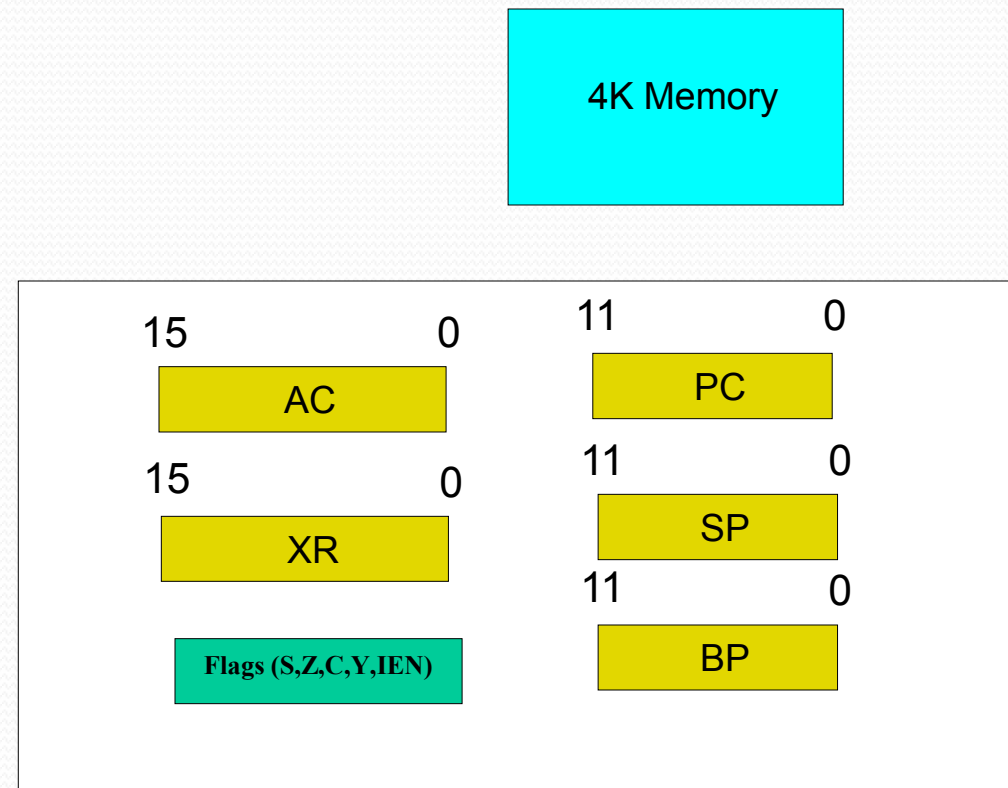
# ISA: instructions

- Memory reference group
  - Bits 12-14 not all ones => memory access and Bits 0-11 are addresses
    - I=1 implies indirect addressing
    - I=0 implies direct addressing
    - total of 7 memory instructions (from 000 to 110)
- Non-memory reference group
  - Bit 12-14 are ones, ops are internal => no need for address. Therefore the Bits 0-11 are used to indicate other instructions

| 000 to 110 | | Address of operand |

| Indicates direct or indirect addressing mode | I  Opcode | 12 bit Address |

**Memory reference instructions**

| X  1  1  1 | Register operations | | Further decodes instructions |

**Non-memory reference instructions**

**111 implies non memory reference**

# Addressing modes

- **direct**
  - bits 0 - 11 is the address of the operand
- **indirect**
  - bits 0 - 11 is the address in which we obtain the address of the operand
    - bit 15 can be used to indicate the addressing mode
  - BP indirect
- **Immediate (not directly supported)**
  - data is placed at a designated location in memory. It can then be loaded using direct or indirect addressing method.
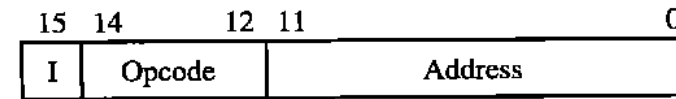


(a) Instruction format
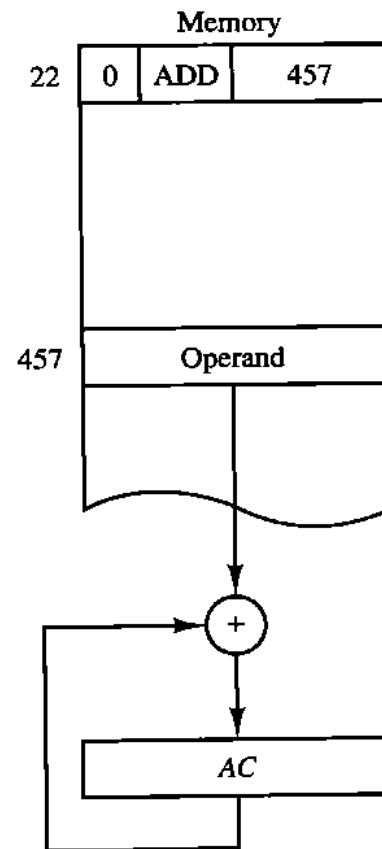
(b) Direct address
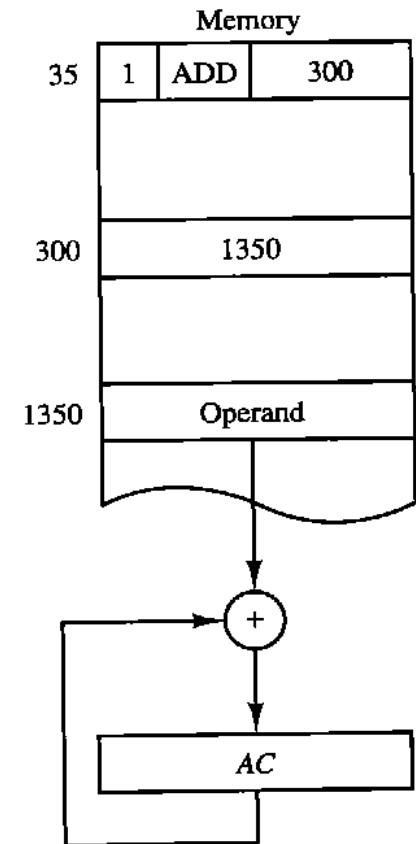
(c) Indirect address

# Definition of Instructions:

Memory reference instructions

| Symbol | Hexadecimal code | | Description |
| --- | --- | --- | --- |
| | I = 0 | I = 1 | |
| LDB | 0xxx | 8xxx | Load memory word to BP |
| LDX | 1xxx | 9xxx | Load memory word to XR |
| LDA | 2xxx | Axxx | Load memory word to AC |
| STA | 3xxx | Bxxx | Store content of AC in memory |
| BUN | 4xxx | Cxxx | Branch unconditionally |
| CALL | 5xxx | Dxxx | Push PC onto stack and branch unconditionally |
| DSZ | 6xxx | Exxx | Decrement and skip if zero |

# Register reference instructions

| | | |
|---|---|---|
| INC | 7010 | Increment AC |
| DEC | 7011 | Decrement AC |
| ADD | 7012 | AC=AC+XR |
| ADC | 7013 | AC=AC+XR+C |
| SUB | 7014 | AC=AC-XR |
| SBB | 7015 | AC=AC-XR+1 |
| AND | 7016 | AC=AC and XR |
| OR | 7017 | AC=AC or XR |
| XOR | 7020 | AC=AC xor XR |
| SHR | 7021 | Shift AC right |
| SHL | 7022 | Shift AC left |
| CLAC | 7023 | Clear AC |
| CMAC | 7024 | Complement AC |
| CLC | 7025 | Clear C |
| CMC | 7026 | Complement C |
| STC | 7027 | Set C |
| AC2SP | 7040 | Move content of AC to SP |
| SP2BP | 7041 | Move content of SP to BP |
| LDABP | 7042 | Load AC indirect through BP |
| IRET | 7043 | Restore address from stack onto PC and ION |
| RET | 7044 | Restore address from stack onto PC |
| POP | 7045 | Pop word from the stack onto AC |
| PUSH | 7046 | Push AC onto the stack |
| SZAC | 7047 | Skip next instruction if AC zero |
| SNAC | 7080 | Skip next instruction if AC negative |
| SPAC | 7081 | Skip next instruction if AC positive |
| SSC | 7082 | Skip next instruction if C is set |
| SCC | 7083 | Skip next instruction if C is clear |
| ION | 7084 | Interrupt on |
| IOF | 7085 | Interrupt off |

arithmetic

logic

Shift

Bit manipulation

Subroutine /stack

Conditional branch

# Instruction set completeness

- Arithmetic
  - add, adc, sub, sbb, inc, dec
  - Instructions like multiple can be synthesized in software
- Logical
  - and, or, not, xor
    - AND and NOT gives NAND => all logic operations can be performed
- Shift instructions
  - shr, shl
- Instructions for moving information to from memory and processor registers
  - lda, sta
- Program control instructions together with instructions that check status condition.
  - bun, call, dsz.  4 skip instructions
- Input and output instructions.
  - I/O devices are memory mapped

# Design of the CPU

The bus

The state generator
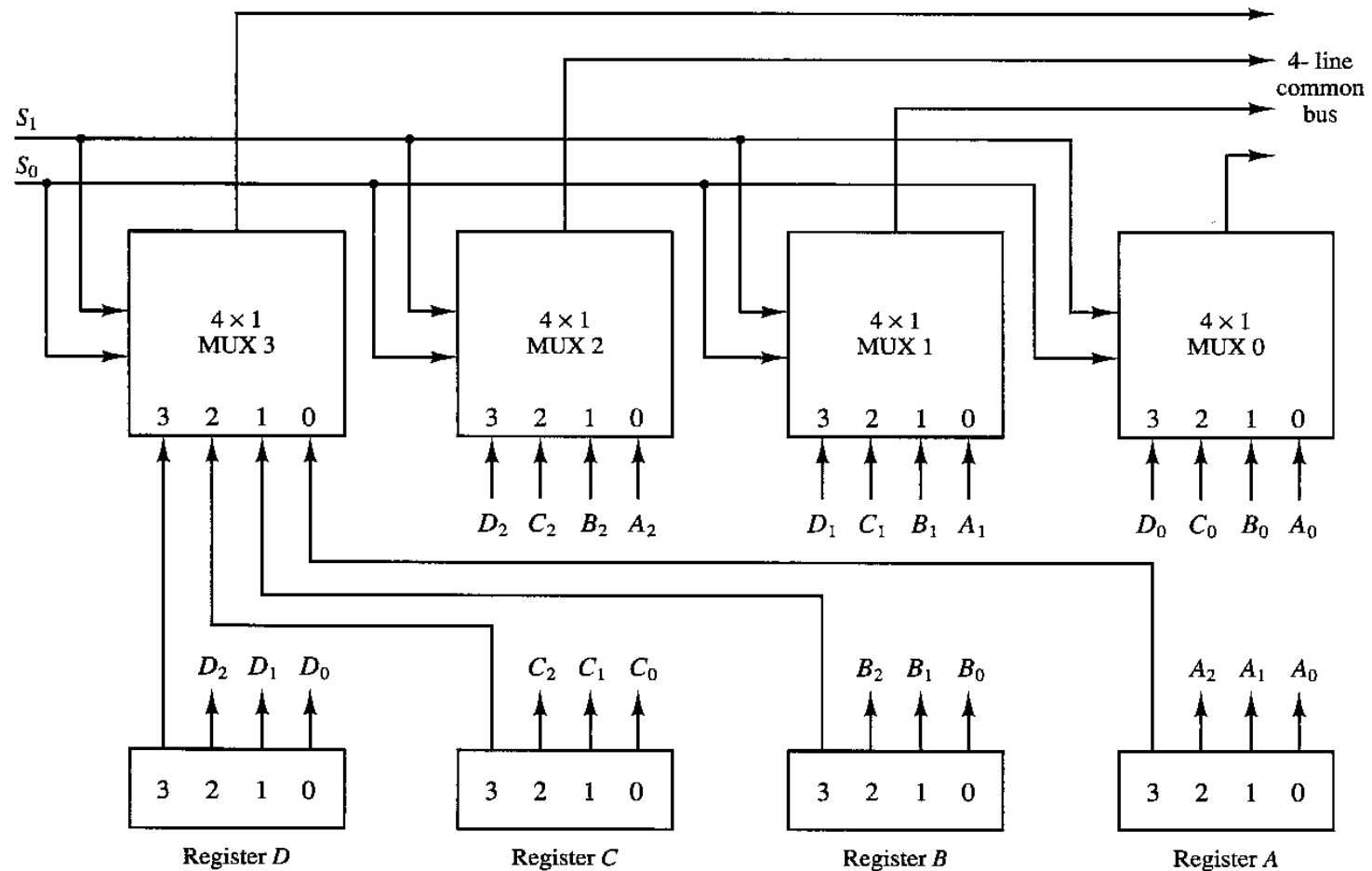
Actions in each state for each instruction

# Common bus concept for transfer of memory content

- Using digital multiplexer
  - intuitive but not normally used in uP design

  - instead we have tristate buffer
    - beside 0 and 1, there is a third high impedance state

    - allows many devices to be electrically tied together as long as only one device asserts a 0 or 1 at any one time

# Construction of a bus system: another perspective

- Bus line with three states buffers Normal input A
  Control input C
  - outputs requires tri-states buffers
  - inputs connected directly to inputs of registers
    - enable to load values
    - check fanout requirement
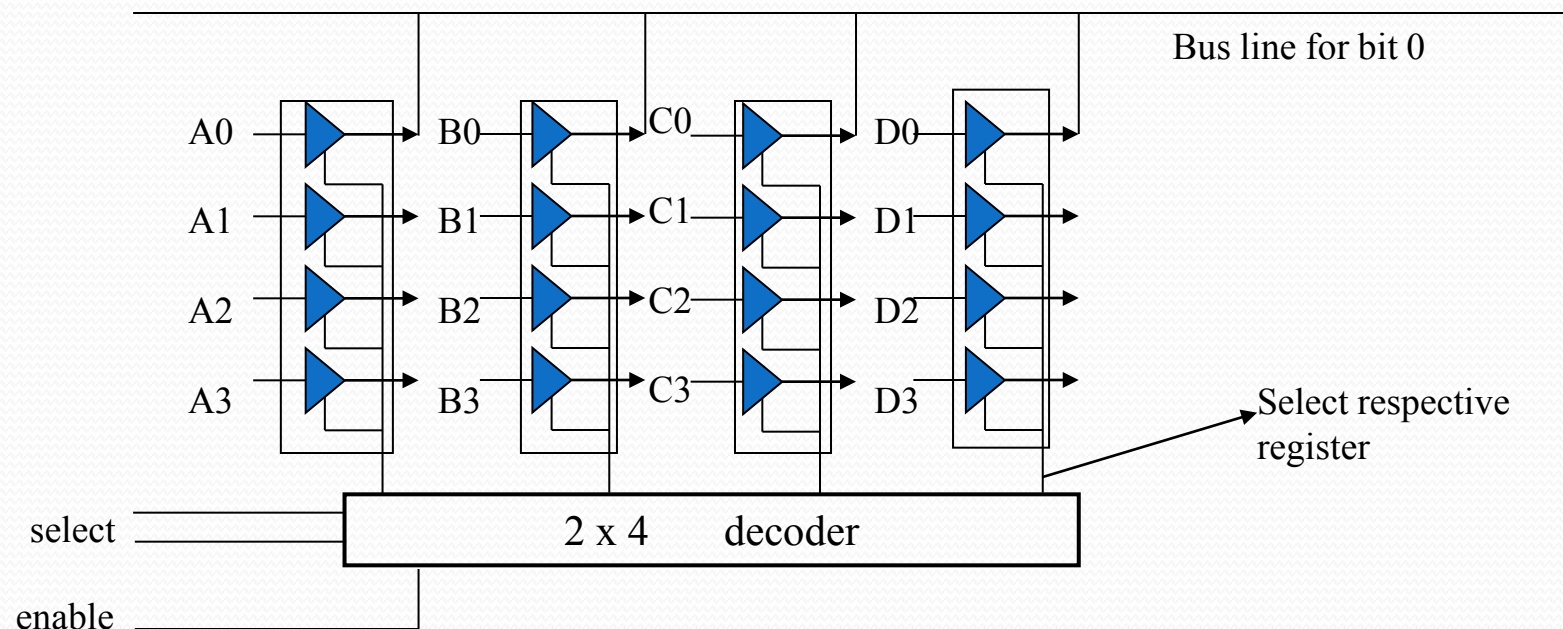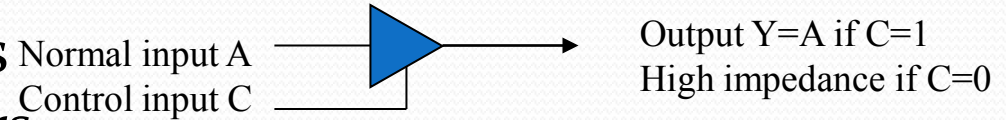
Output Y=A if C=1
High impedance if C=0

Bus line for bit 0

A0    B0    C0    D0

A1    B1    C1    D1

A2    B2    C2    D2

A3    B3    C3    D3

Select respective register

select

2 x 4     decoder

enable

# ISA: Registers and memory overview

- PC: program counter
  - Holds address of the next instruction
- AR: address register
  - holds address for selecting memory
- SP: stack pointer
- BP: base pointer
- IR: instruction register
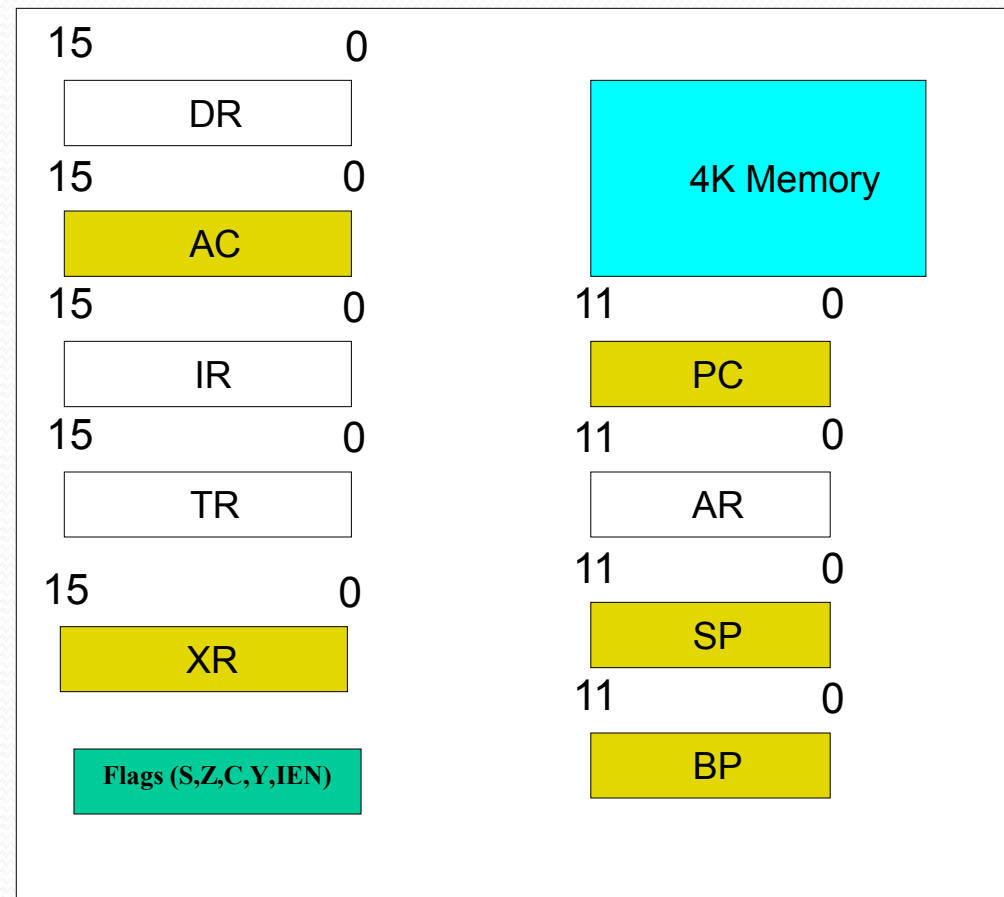  - holds the instruction code for decoding
- TR: temporary register
- DR: data register
  - holds memory operand
- AC: accumulator
  - main processor register
- XR: operand register
  - For holding the operand of arithmetic/logic operations
- Flags
  - S=AC negative
  - Z=AC zero
  - C=carry flag set
  - Y=DR=o

**What are the objects in the system?**
**What are available to the programmer?**

| 15 | 0 |
| --- | --- |
| DR | |

| 15 | 0 |
| --- | --- |
| AC | |

4K Memory

| 15 | 0 |
| --- | --- |
| IR | |

| 11 | 0 |
| --- | --- |
| PC | |

| 15 | 0 |
| --- | --- |
| TR | |

| 11 | 0 |
| --- | --- |
| AR | |

| 15 | 0 |
| --- | --- |
| XR | |

| 11 | 0 |
| --- | --- |
| SP | |

Flags (S,Z,C,Y,IEN)

| 11 | 0 |
| --- | --- |
| BP | |

# Common bus system of uP4202

- 11 items
  - internal bus to control the movement of data among them
  - 4 bit control lines to determine who drives bus
  - LD line to load specific register
  - the specific word location in the 4K memory is selected by AR
- Control
  - We need to generate S0-S3, LD, INR, CLR to control the movement of data around at the appropriate time.

S0
S1
S2
S3

BUS

Memory Unit
4096 X 16

ADDRESS

Write          Read

7

AR

LD     INR     CLR

1

PC

LD     INR     CLR

2

DR

LD     INR     CLR

3

ADDER
AND
LOGIC

E

AC

LD     INR     CLR     DEC

4

IR

LD

5

TR

LD     INR     CLR

6

XR

LD     INR     CLR

11

FLAGS

10

SP

LD     INR     DEC

8

BP

LD     INR     CLR

9

Clock
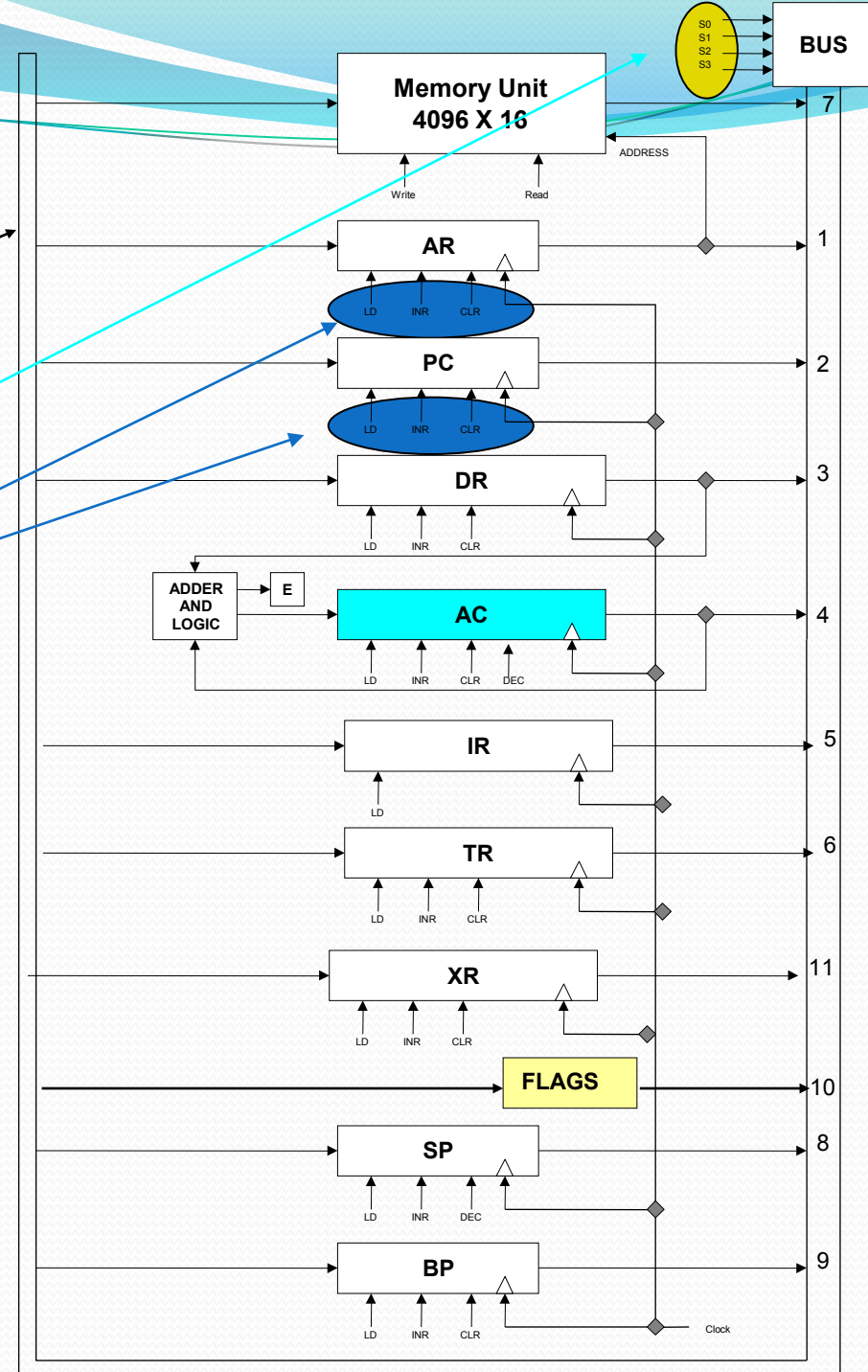
# Timing and control

- Creation of sequencing control in the decoding of instructions

**State generator**

Instructional Register

| 15 | 14 13 12 | 11 - 0 |
|---|---|---|

3 x 8
Decoder

7 6 5 4 3 2 1 0

I

Other Inputs

$D_0$

$D_7$

Control Logic Gates

Control Output

*Design this….*

$T_{15}$

$T_0$

| 15 14 ..........2 1 0 | |
|---|---|

4 x 16
SC

Inc
Clear
Clock

**Tell what need to be done.**

**State signals – tell when to do something. SC is cleared as and when necessary**

Clock
$T_0$
$T_1$
$T_2$
$T_3$
$T_4$
$D_3$
CLR
SC

$T_0$  $T_1$  $T_2$  $T_3$  $T_4$  $T_0$

# Instruction cycle

- Fetch an instruction from memory
- Decode instruction
- Read the effective address from if the instruction has an indirect address
- execute the instruction

**An instruction always start with T0, then T1, ....
During states T0 and T1, instruction is fetched.
How to fetch an instruction?**
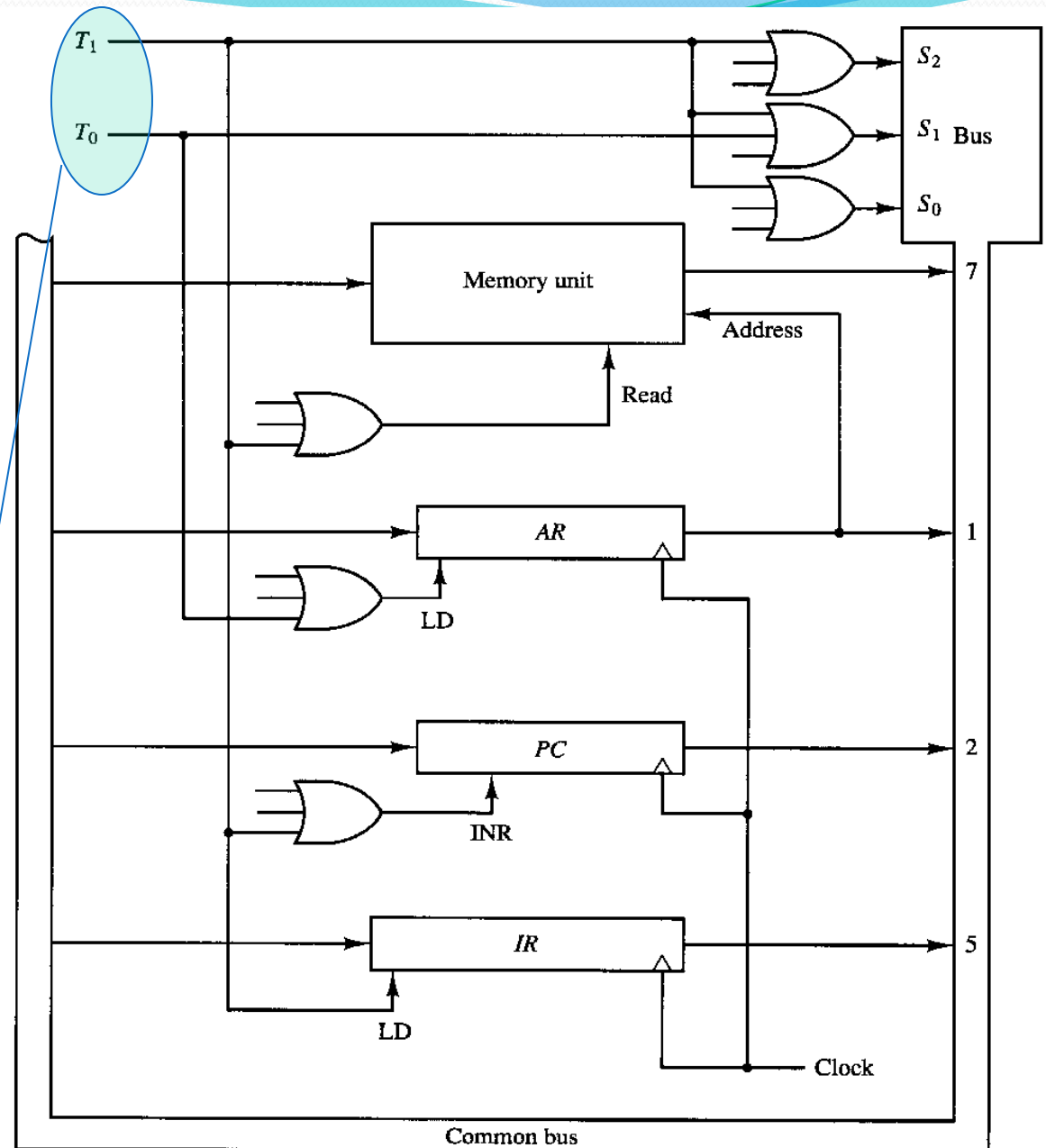
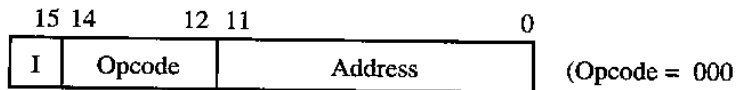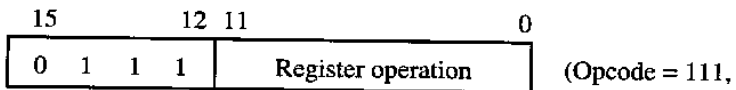# During State T2:

- **DECODE.** Set
  - D7←1 if opcode =111
  - I ← IR(15)
  - AR ← IR(0-11)



(a) Memory – reference instruction

| 15 | 14 | | 12 | 11 | | 0 |
|----|----|----|----|----|----|---|
| I | Opcode | | | Address | | |

(Opcode = 000

(b) Register – reference instruction

| 15 | | | 12 | 11 | | 0 |
|----|----|----|----|----|----|---|
| 0 | 1 | 1 | 1 | Register operation | | |

(Opcode = 111,
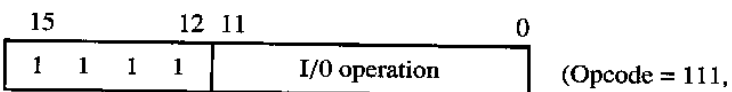
(c) Input – output instruction

| 15 | | | 12 | 11 | | 0 |
|----|----|----|----|----|----|---|
| 1 | 1 | 1 | 1 | I/0 operation | | |

(Opcode = 111,



Start
SC ← 0

$T_0$
AR ← PC

$T_1$
IR ← M [AR], PC ← PC + 1

$T_2$
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)

=1 non-memory    $D_7$    = 0 (Memory-reference)

(indirect) = 1    I    = 0 (direct)

$T_3$
Execute register-reference instruction
SC ← 0

$T_3$
AR ← M[AR]

$T_3$
Nothing

Execute memory-reference instruction
SC ← 0
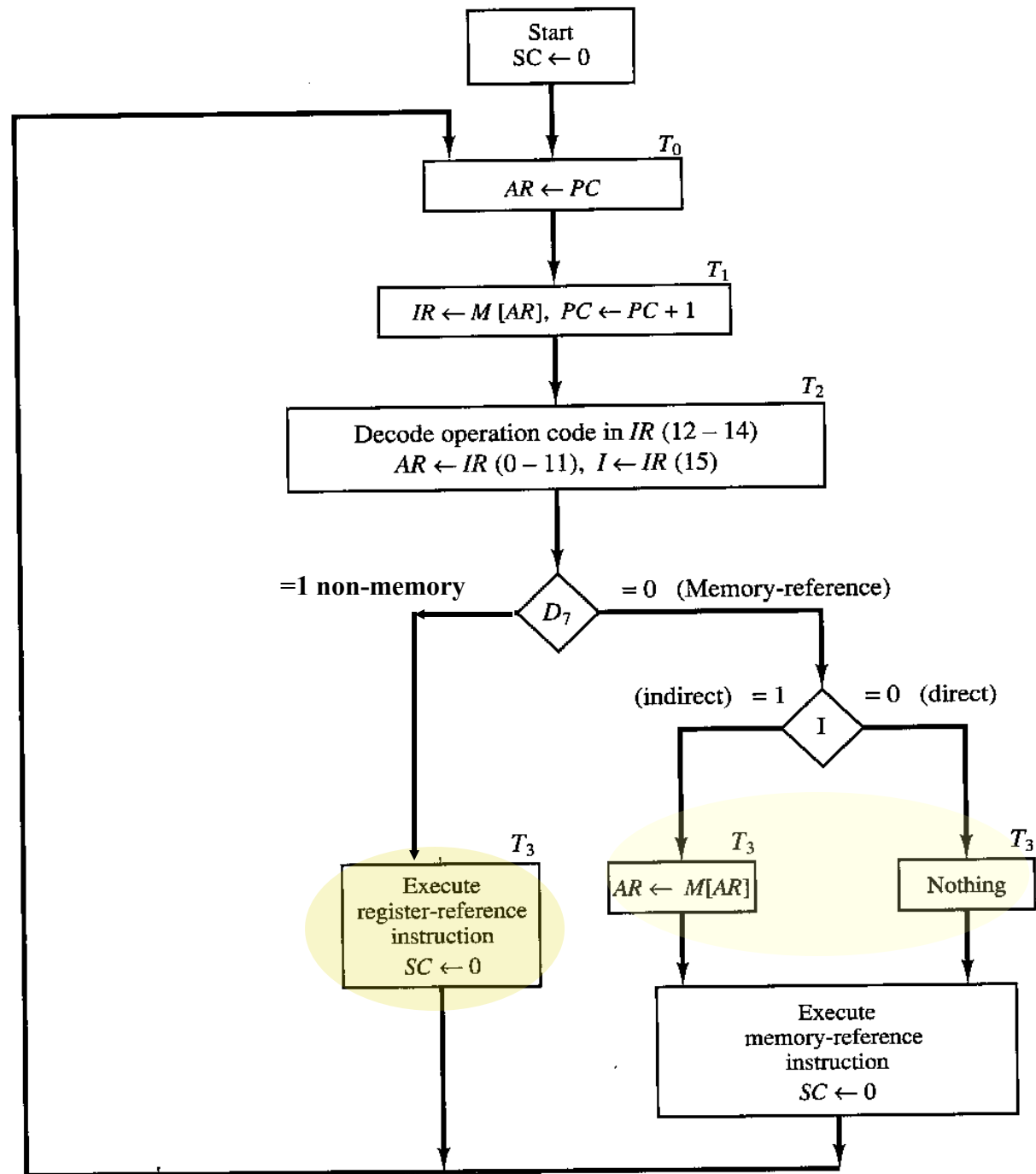
# Memory reference instructions

- Do to D6 determine each one of the below instruction
- from flowchart effective address placed during T2 or T3 depending on whether direct or indirect addressing

| | | | | | |
|---|---|---|---|---|---|
| LDB | 0XXX | 8XXX | $D_0T_4$ | $BP \leftarrow M[AR], SC \leftarrow 0$ | Bus transfer |
| LDX | 1XXX | 9XXX | $D_1T_4$ | $XR \leftarrow M[AR], SC \leftarrow 0$ | Bus transfer |
| LDA | 2XXX | AXXX | $D_2T_4$<br>$D_2T_5$ | $DR \leftarrow M[AR], AC \leftarrow 0$<br>$AC \leftarrow AC + DR, SC \leftarrow 0$ | Note that input to AC is connected to ALU, not bus. |
| STA | 3XXX | BXXX | $D_3T_4$ | $M[AR] \leftarrow AC, SC \leftarrow 0$ | Bus transfer |
| BUN | 4XXX | CXXX | $D_4T_4$ | $PC \leftarrow AR, SC \leftarrow 0$ | Branch address already in AR by $T_3$ |
| CALL | 5XXX | DXXX | $D_5T_4$<br>$D_5T_5$<br>$D_5T_6$<br>$D_5T_7$ | $SP \leftarrow SP - 1, TR \leftarrow AR$<br>$AR \leftarrow SP$<br>$M[AR] \leftarrow PC$<br>$PC \leftarrow TR, SC \leftarrow 0$ | Store in TR the branch address that is in AR. Store current PC on stack. Transfer TR to PC to branch. |
| DSZ | 6XXX | EXXX | $D_6T_4$<br>$D_6T_5$<br>$D_6T_6Y$<br>$D_6T_6Y'$ | $DR \leftarrow M[AR]$<br>$DR \leftarrow DR - 1$<br>$M[AR] \leftarrow DR, PC \leftarrow PC + 1, SC \leftarrow 0$<br>$M[AR] \leftarrow DR, SC \leftarrow 0$ | Read in data to DR.<br>Check for zero to branch |

# Register reference instruction in T3

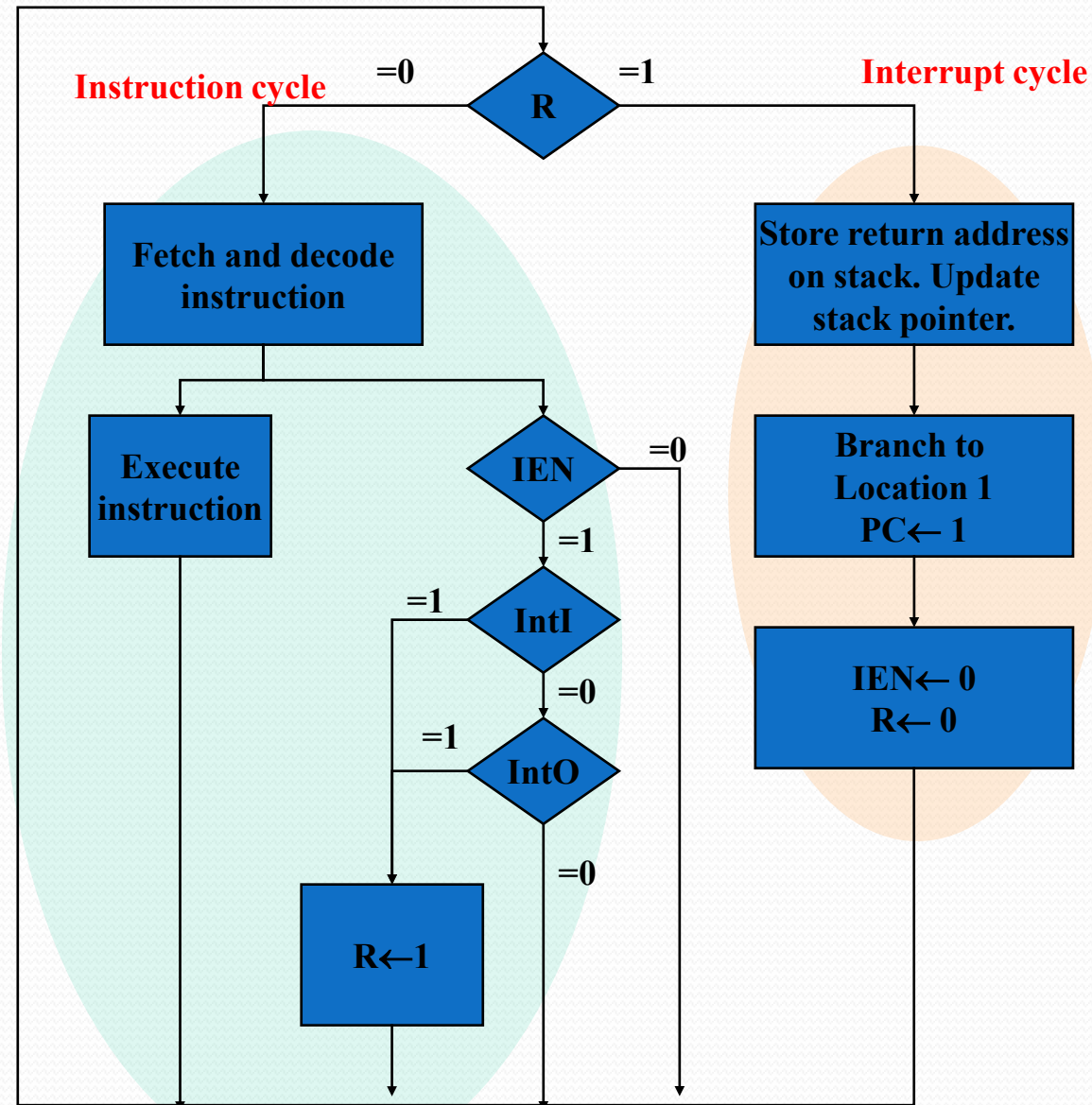| | | | $D_7I'\ T_3 = r$<br>$IR(i) = Bi\ (i=0\ TO\ 11)$<br>$K_0=B_2'B_1'B_0',\ K_1=B_2'B_1'B_0,\ ...,\ K_7=B_2B_1B_0,$ | |
|------|------|------|------|------|
| | | r | $SC \leftarrow 0$ | |
| INC | 7010 | $rB_4K_0$ | $AC \leftarrow AC + 1,\ SC \leftarrow 0$ | Increment AC |
| DEC | 7011 | $rB_4K_1$ | $AC \leftarrow AC-1,\ SC \leftarrow 0$ | Decrement AC |
| ADD | 7012 | $D_7I'\ T_3\ B_4K_2$<br>$D_7I'\ T_4\ B_4K_2$ | $DR \leftarrow XR$<br>$AC \leftarrow AC + DR,\ SC \leftarrow 0$ | AC=AC+XR |
| ADC | 7013 | $D_7I'\ T_3\ B_4K_3$<br>$D_7I'\ T_4\ B_4K_3$ | $DR \leftarrow XR$<br>$AC \leftarrow AC + DR +C,\ SC \leftarrow 0$ | AC=AC+XR+C |
| SUB | 7014 | $D_7I'\ T_3\ B_4K_4$<br>$D_7I'\ T_4\ B_4K_4$ | $DR \leftarrow XR$<br>$AC \leftarrow AC - DR,\ SC \leftarrow 0$ | AC=AC-XR |
| SBB | 7015 | $D_7I'\ T_3\ B_4K_5$<br>$D_7I'\ T_4\ B_4K_5$ | $DR \leftarrow XR$<br>$AC \leftarrow AC - DR +C,\ SC \leftarrow 0$ | AC=AC-XR+1 |
| AND | 7016 | $D_7I'\ T_3\ B_4K_6$<br>$D_7I'\ T_4\ B_4K_6$ | $DR \leftarrow XR$<br>$AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$ | AC=AC and XR |
| OR | 7017 | $D_7I'\ T_3\ B_4K_7$<br>$D_7I'\ T_4\ B_4K_7$ | $DR \leftarrow XR$<br>$AC \leftarrow AC \vee DR,\ SC \leftarrow 0$ | AC=AC or XR |
| XOR | 7020 | $D_7I'\ T_3\ B_5K_0$<br>$D_7I'\ T_4\ B_5K_0$ | $DR \leftarrow XR$<br>$AC \leftarrow AC \oplus DR,\ SC \leftarrow 0$ | AC=AC xor XR |
| SHR | 7021 | $rB_5K_1$ | $AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0),\ SC \leftarrow 0$ | Shift AC right |
| SHL | 7022 | $rB_5K_2$ | $AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15),\ SC \leftarrow 0$ | Shift AC left |

# Register reference instruction in T3

| CLAC | 7023 | $rB_5K_3$ | $AC \leftarrow 0$ , $SC \leftarrow 0$ | Clear AC |
|------|------|-----------|-------------------------------|----------|
| CMAC | 7024 | $rB_5K_4$ | $AC \leftarrow AC'$ , $SC \leftarrow 0$ | Complement AC |
| CLC | 7025 | $rB_5K_5$ | $C \leftarrow 0$ , $SC \leftarrow 0$ | Clear C |
| CMC | 7026 | $rB_8K_6$ | $C \leftarrow C'$ , $SC \leftarrow 0$ | Complement C |
| STC | 7027 | $rB_5K_7$ | $C \leftarrow 1$ , $SC \leftarrow 0$ | Set C |
| AC2SP | 7040 | $rB_6K_0$ | $SP \leftarrow AC$, $SC \leftarrow 0$ | Move content of AC to SP |
| SP2BP | 7041 | $rB_6K_1$ | $BP \leftarrow SP$, $SC \leftarrow 0$ | Move content of SP to BP |
| LDABP | 7042 | $D_7I' \, T_3 \, B_6K_2$ <br> $D_7I' \, T_4 \, B_6K_2$ <br> $D_7I' \, T_5 \, B_6K_2$ | $AR \leftarrow BP$, $AC \leftarrow 0$ <br> $DR \leftarrow M[AR]$ <br> $AC \leftarrow AC + DR$, $SC \leftarrow 0$ | Load AC indirect through BP |

# Register reference instruction in T3

| | | | | |
|---|---|---|---|---|
| IRET | 7043 | $D_7I'T_3B_6K_3$<br>$D_7I'T_4B_6K_3$<br>$D_7I'T_5B_6K_3$<br>$D_7I'T_6B_6K_3$ | $AR \leftarrow SP$<br>$PC \leftarrow M[AR], SP \leftarrow SP+1$<br>$AR \leftarrow SP$<br>$FLAGS \leftarrow M[AR], SP \leftarrow SP+1, IEN$<br>$\leftarrow 1, SC \leftarrow 0$ | Restore address from stack onto PC and ION |
| RET | 7044 | $D_7I'T_3B_6K_4$<br>$D_7I'T_4B_6K_4$ | $AR \leftarrow SP$<br>$PC \leftarrow M[AR], SP \leftarrow SP+1, SC \leftarrow 0$ | Restore address from stack onto PC |
| POP | 7045 | $D_7I'T_3B_6K_5$<br>$D_7I'T_4B_6K_5$ | $AR \leftarrow SP, AC \leftarrow 0$<br>$DR \leftarrow M[AR], SP \leftarrow SP+1$<br>$AC \leftarrow AC+DR, SC \leftarrow 0$ | Pop word from the stack onto AC |
| PUSH | 7046 | $D_7I'T_3B_6K_6$<br>$D_7I'T_4B_6K_6$<br>$D_7I'T_5B_6K_6$ | $SP \leftarrow SP-1$<br>$AR \leftarrow SP$<br>$M[AR] \leftarrow AC, SC \leftarrow 0$ | Push AC onto the stack |
| SZAC | 7047 | $rB_6K_7Z$<br>$rB_6K_7Z'$ | $PC \leftarrow PC+1, SC \leftarrow 0$<br>$SC \leftarrow 0$ | Skip next instruction if AC zero |
| SNAC | 7080 | $rB_7K_0S$<br>$rB_7K_0S'$ | $PC \leftarrow PC+1, SC \leftarrow 0$<br>$SC \leftarrow 0$ | Skip next instruction if AC negative |
| SPAC | 7081 | $rB_7K_1S'Z'$<br>$rB_7K_1(S'Z')'$ | $PC \leftarrow PC+1, SC \leftarrow 0$<br>$SC \leftarrow 0$ | Skip next instruction if AC positive |
| SSC | 7082 | $rB_7K_2C$<br>$rB_7K_2C'$ | $PC \leftarrow PC+1, SC \leftarrow 0$<br>$SC \leftarrow 0$ | Skip next instruction if C is set |
| SCC | 7083 | $rB_7K_3C'$<br>$rB_7K_3C$ | $PC \leftarrow PC+1, SC \leftarrow 0$<br>$SC \leftarrow 0$ | Skip next instruction if C is clear |
| ION | 7084 | $rB_7K_4$ | $IEN \leftarrow 1, SC \leftarrow 0$ | Interrupt on |
| IOF | 7085 | $rB_7K_5$ | $IEN \leftarrow 0, SC \leftarrow 0$ | Interrupt off |

# Interrupts

- R is an interrupt indicator flag
- request for I/O will interrupt uP
  - Int1=1 or Int0=1
  - Both signals are externally generated.
  - Signals tie to two D f/fs.
  - At each clock cycle, status is updated.
- During instruction execution, flags will be check and R set or clear accordingly
- R is check before every instruction to determine if uP should go into interrupt cycle or not
- Interrupt vector address at location 1

**Instruction cycle**

**Interrupt cycle**

R  =0 / =1

Fetch and decode instruction

Execute instruction

IEN  =1  =0

IntI  =1  =0

IntO  =1  =0

R←1

Store return address on stack. Update stack pointer.

Branch to Location 1
PC← 1

IEN← 0
R← 0

# Interrupt cycles

- Log the interrupt event during instruction cycle.
  - $T_0' T_1' T_2'$ (IEN)(INT0+INT1): $R \leftarrow 1$
- Interrupt occurs <span style="color:red">asynchronously</span>. Need to save
  - Return address
  - Status of current execution
    - Eg: interrupt occurs between an arithmetic instruction that set a flag and conditional jump instruction that acts on that flag.
  - Determine the interrupt vector address and branch to that address
  - Disable further interrupts (clear IEN flag).
    - User may re-enable the interrupt in the interrupt service routine.
- When R is set, go into interrupt acknowledgement cycle
  - $RT_0$      : SP←SP-1, TR←PC
  - $RT_1$      : AR ←SP
  - $RT_2$      : M[AR]←FLAGS, SP ←SP-1
  - $RT_3$      : AR ←SP, PC←0
  - $RT_4$      : M[AR]←TR, PC←PC+1, IEN←0, R←0, SC←0

Interrupt vector address is at location 1

# Listing of all control functions and microoperations

| Fetch | R'T0 | : AR←PC |
|---|---|---|
| | R'T1 | : IR←M[AR], PC←PC+1 |
| Decode | R'T2 | : D0,…, D7←Decode IR(12-14), AR←IR(0-11),  I←IR(15) |
| Indirect | D7'IT3 | : AR←M[AR] |
| Interrupt | T0'T1'T2'(IEN)(FGI+FGO) | : R←1 |
| Interrupt decode | RT0 | : SP←SP-1, TR←PC |
| | RT1 | : AR ←SP |
| | RT2 | : M[AR]←FLAGS, SP ←SP-1 |
| | RT3 | : AR←SP, PC←0 |
| | RT4 | : M[AR]←TR, PC←PC+1, IEN←0, R←0, SC←0 |
| Memory Reference Instruction (see table) | | |
| Register Reference Instruction (see table) | | |
| Initialization: Reset  RS=1   : clear all  (RS= D F/F with D input connected to reset pin. As long as D is high, Q output will remain high. | | |

# Synthesizing the Control logic

- Signals for S3,S2, S1 and S0 to select which of the 11 registers drive the bus
- Signals to control the inputs (LD,INR,CLR,DEC) of the 11 registers
- Signals to control the read and write inputs of memory
- Signals to set, clear, or complement the bit addressable FF
- Signals to control the ALU circuit

**Interpreting RTL into logic gates**

Load the content on the bus into AR.

When this condition occurs, perform the operation on the right.

## RT1: AR ←SP

Content of SP to be placed on the bus.

# Select which of the 10 registers drive the bus

| | | | |
|---|---|---|---|
| BUN | $D_4T_4$ | PC ← AR | X1: AR |
| CALL | $D_5T_4$ | TR ← AR | |
| Fetch | $R'T_0$ | AR ← PC | X2: PC |
| Interrupt | $RT_0$ | TR ← PC | |
| CALL | $D_5T_6$ | M[AR] ← PC | |
| DSZ | $D_6T_6$ | M[AR] ← DR | X3: DR |
| STA | $D_3T_4$ | M[AR] ← AC | X4: AC |
| PUSH | $D_7I'\ T_5 B_6 K_6$ | M[AR] ← AC | |
| AC2SP | $rB_6K_0$ | SP ← AC | |
| Decode | $R'T_2$ | AR ← IR(0-11) | X5: IR |
| Interrupt | $RT_4$ | M[AR]←TR | X6: TR |
| CALL | $D_5T_7$ | PC ← TR | |
| Fetch | $R'T_1$ | IR ← M[AR] | X7: Memory |
| Indirect | $D_7'T_3$ | AR ← M[AR] | |
| LDB | $D_0T_4$ | BP ← M[AR] | |
| LDX | $D_1T_4$ | XR ← M[AR] | |
| LDA | $D_2T_4$ | DR ← M[AR] | |
| DSZ | $D_6T_4$ | DR ← M[AR] | |
| LDABP | $D_7I'\ T_4 B_6 K_2$ | DR ← M[AR] | |
| IRET | $D_7I'\ T_4 B_6 K_3$ | PC ← M[AR] | |
| IRET | $D_7I'\ T_6 B_6 K_3$ | FLAGS ← M[AR] | |
| RET | $D_7I'\ T_4 B_6 K_4$ | PC ← M[AR] | |
| POP | $D_7I'\ T_4 B_6 K_5$ | DR ← M[AR] | |

**X8=OR(…)**

| | | | |
|---|---|---|---|
| Interrupt | $RT_1$ | AR ← SP | X8: SP |
| Interrupt | $RT_3$ | AR ←SP | |
| CALL | $D_5T_5$ | AR ← SP | |
| SP2BP | $rB_6K_1$ | BP ← SP | |
| IRET | $D_7I'\ T_3 B_6 K_3$ | AR ← SP | |
| IRET | $D_7I'\ T_5 B_6 K_3$ | AR ←SP | |
| RET | $D_7I'\ T_3 B_6 K_4$ | AR ← SP | |
| POP | $D_7I'\ T_3 B_6 K_5$ | AR ← SP | |
| PUSH | $D_7I'\ T_4 B_6 K_6$ | AR ← SP | |
| LDABP | $D_7I'\ T_3 B_6 K_2$ | AR ← BP | X9: BP |
| Interrupt | $RT_2$ | M[AR]←FLAGS | X10: FLAGS |
| ADD | $D_7I'\ T_3 B_4 K_2$ | DR←XR | X11: XR |
| ADC | $D_7I'\ T_3 B_4 K_3$ | DR←XR | |
| SUB | $D_7I'\ T_3 B_4 K_4$ | DR←XR | |
| SBB | $D_7I'\ T_3 B_4 K_5$ | DR←XR | |
| AND | $D_7I'\ T_3 B_4 K_6$ | DR←XR | |
| OR | $D_7I'\ T_3 B_4 K_7$ | DR←XR | |
| XOR | $D_7I'\ T_3 B_5 K_0$ | DR←XR | |

# Control of common bus via S0,S1,S2,S3

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | $x_{10}$ | $x_{11}$ | $S_3$ | $S_2$ | $S_1$ | $S_0$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | None |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | AR |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | PC |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | DR |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | AC |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | IR |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | TR |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | MEMORY |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | SP |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | BP |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | FLAGS |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | XR |

- From table of control functions, pick out all functions requiring each register to drive bus
- Derive logic function for X
  - Xn=OR (corresponding conditions )
- Send Xn into a 16 to 4 encoder to derive S0, S1, S2, S3.
- S0, S1, S2, S3 are then used to drive the bus selectors.

## Signals to control the (LD,INR,CLR,DEC) of the 11 registers

| | | |
|---|---|---|
| RET | $D_7 I' T_3 B_6 K_4$ | AR ← SP |
| IRET | $D_7 I' T_3 B_6 K_3$ | AR ← SP |
| CALL | $D_5 T_5$ | AR ← SP |
| PUSH | $D_7 I' T_4 B_6 K_6$ | AR ← SP |
| POP | $D_7 I' T_3 B_6 K_5$ | AR ← SP |
| Interrupt | $RT_3$ | AR ← SP |
| Interrupt | $RT_1$ | AR ← SP |
| IRET | $D_7 I' T_5 B_6 K_3$ | AR ← SP |
| Fetch | $R'T_0$ | AR ← PC |
| LDABP | $D_7 I' T_3 B_6 K_2$ | AR ← BP |
| Indirect | $D_7' T_3$ | AR ← M[AR] |
| Decode | $R'T_2$ | AR ← IR(0-11) |
| Interrupt | $RT_3$ | PC ← 0 |
| Interrupt | $RT_4$ | PC ← PC+1 |
| SZAC | $rB_6 K_7 Z$ | PC ← PC + 1 |
| SPAC | $rB_7 K_1 S'Z'$ | PC ← PC + 1 |
| SNAC | $rB_7 K_0 S$ | PC ← PC + 1 |
| SCC | $rB_7 K_3 C'$ | PC ← PC + 1 |
| DSZ | $D_6 T_6 Y$ | PC ← PC + 1 |
| Fetch | $R'T_1$ | PC ← PC + 1 |
| IRET | $D_7 I' T_4 B_6 K_3$ | PC ← M[AR] |
| RET | $D_7 I' T_4 B_6 K_4$ | PC ← M[AR] |
| CALL | $D_5 T_7$ | PC ← TR |
| BUN | $D_4 T_4$ | PC ← AR |

**CLR$_{PC}$ = RT$_4$ ∨ reset**

**INR$_{PC}$ = OR( … )**

**LD$_{PC}$ = OR( … )**

**Group all records with PC on the left. Implies that under the conditions, PC is to be changed accordingly.**

# More controls

| | | |
|---|---|---|
| PUSH | $D_7I'\,T_5\,B_6K_6$ | $M[AR] \leftarrow AC$ |
| STA | $D_3T_4$ | $M[AR] \leftarrow AC$ |
| Interrupt | $RT_4$ | $M[AR] \leftarrow TR$ |
| Interrupt | $RT_2$ | $M[AR] \leftarrow FLAGS$ |
| CALL | $D_5T_6$ | $M[AR] \leftarrow PC$ |
| DSZ | $D_6T_6$ | $M[AR] \leftarrow DR$ |
| Interrupt | $RT_0$ | $TR \leftarrow PC$ |
| CALL | $D_5T_4$ | $TR \leftarrow AR$ |
| LDX | $D_1T_4$ | $XR \leftarrow M[AR]$ |
| DSZ | $D_6T_5$ | $DR \leftarrow DR - 1$ |
| LDABP | $D_7I'\,T_4\,B_6K_2$ | $DR \leftarrow M[AR]$ |
| POP | $D_7I'\,T_4\,B_6K_5$ | $DR \leftarrow M[AR]$ |
| LDA | $D_2T_4$ | $DR \leftarrow M[AR]$ |
| DSZ | $D_6T_4$ | $DR \leftarrow M[AR]$ |
| XOR | $D_7I'\,T_3\,B_5K_0$ | $DR \leftarrow XR$ |
| ADD | $D_7I'\,T_3\,B_4K_2$ | $DR \leftarrow XR$ |
| ADC | $D_7I'\,T_3\,B_4K_3$ | $DR \leftarrow XR$ |
| SBB | $D_7I'\,T_3\,B_4K_5$ | $DR \leftarrow XR$ |
| SUB | $D_7I'\,T_3\,B_4K_4$ | $DR \leftarrow XR$ |
| AND | $D_7I'\,T_3\,B_4K_6$ | $DR \leftarrow XR$ |
| OR | $D_7I'\,T_3\,B_4K_7$ | $DR \leftarrow XR$ |

| | | |
|---|---|---|
| CALL | $D_5T_4$ | $SP \leftarrow SP - 1$ |
| PUSH | $D_7I'\,T_3\,B_6K_6$ | $SP \leftarrow SP - 1$ |
| Interrupt | $RT_2$ | $SP \leftarrow SP - 1$ |
| Interrupt | $RT_0$ | $SP \leftarrow SP - 1$ |
| RET | $D_7I'\,T_4\,B_6K_4$ | $SP \leftarrow SP + 1$ |
| IRET | $D_7I'\,T_4\,B_6K_3$ | $SP \leftarrow SP + 1$ |
| IRET | $D_7I'\,T_6\,B_6K_3$ | $SP \leftarrow SP + 1$ |
| POP | $D_7I'\,T_4\,B_6K_5$ | $SP \leftarrow SP + 1$ |
| AC2SP | $rB_6K_0$ | $SP \leftarrow AC$ |
| SP2BP | $rB_6K_1$ | $BP \leftarrow SP$ |
| LDB | $D_0T_4$ | $BP \leftarrow M[AR]$ |
| Fetch | $R'T_1$ | $IR \leftarrow M[AR]$ |
| IRET | $D_7I'\,T_6\,B_6K_3$ | $FLAGS \leftarrow M[AR]$ |

# Logic for control of F/F

| | | |
|---|---|---|
| CLC | $rB_5K_5$ | $C \leftarrow 0$ |
| CMC | $rB_8K_6$ | $C \leftarrow C'$ |
| STC | $rB_5K_7$ | $C \leftarrow 1$ |
| IRET | $D_7I'\,T_6\,B_6K_3$ | $IEN \leftarrow 1$ |
| ION | $rB_7K_4$ | $IEN \leftarrow 1$ |
| IOF | $rB_7K_5$ | $IEN \leftarrow 0$ |
| Interrupt | $RT_4$ | $IEN \leftarrow 0$ |
| Interrupt | $RT_4$ | $R \leftarrow 0$ |
| Interrupt | T0'T1'T2'(IEN(FGI +FGO) | $R \leftarrow 1$ |
| Decode | $R'T_2$ | $I \leftarrow IR(15)$ |

**Link to ALU. To be considered with ALU design**

rB7
$K_4$
ION

$D_7$
I'
$T_6$
$B_6$
$K_3$
IRET

$K_5$
IOF

R
$T_4$
INTERRUPT

J    Q    IEN

Clock

K    $\overline{Q}$
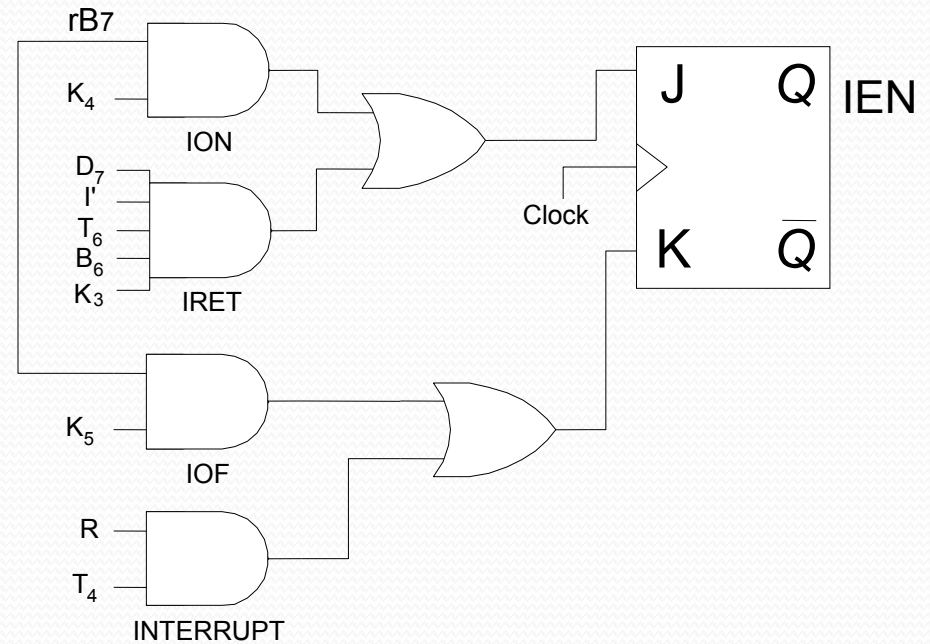
## Circuit for IEN F/F

# Completing the picture for the F/F

| | | |
|---|---|---|
| SHR | $rB_5K_1$ | $SC \leftarrow 0$ |
| SHL | $rB_5K_2$ | $SC \leftarrow 0$ |
| CLAC | $rB_5K_3$ | $SC \leftarrow 0$ |
| IRET | $D_7I'\ T_6\,B_6K_3$ | $SC \leftarrow 0$ |
| LDABP | $D_7I'\ T_5\,B_6K_2$ | $SC \leftarrow 0$ |
| CMAC | $rB_5K_4$ | $SC \leftarrow 0$ |
| CLC | $rB_5K_5$ | $SC \leftarrow 0$ |
| CMC | $rB_8K_6$ | $SC \leftarrow 0$ |
| STC | $rB_5K_7$ | $SC \leftarrow 0$ |
| AC2SP | $rB_6K_0$ | $SC \leftarrow 0$ |
| SP2BP | $rB_6K_1$ | $SC \leftarrow 0$ |
| BUN | $D_4T_4$ | $SC \leftarrow 0$ |
| LDA | $D_2T_5$ | $SC \leftarrow 0$ |
| LDB | $D_0T_4$ | $SC \leftarrow 0$ |
| LDX | $D_1T_4$ | $SC \leftarrow 0$ |
| Interrupt | $RT_4$ | $SC \leftarrow 0$ |
| CALL | $D_5T_7$ | $SC \leftarrow 0$ |
| DSZ | $D_6T_6$ | $SC \leftarrow 0$ |
| INC | $rB_4K_0$ | $SC \leftarrow 0$ |
| DEC | $rB_4K_1$ | $SC \leftarrow 0$ |
| ADD | $D_7I'\ T_4\,B_4K_2$ | $SC \leftarrow 0$ |
| STA | $D_3T_4$ | $SC \leftarrow 0$ |

| | | |
|---|---|---|
| RET | $D_7I'\ T_4\,B_6K_4$ | $SC \leftarrow 0$ |
| IOF | $rB_7K_5$ | $SC \leftarrow 0$ |
| ION | $rB_7K_4$ | $SC \leftarrow 0$ |
| SCC | $rB_7K_3C'$ | $SC \leftarrow 0$ |
| SCC | $rB_7K_3C$ | $SC \leftarrow 0$ |
| SPAC | $rB_7K_1$ | $SC \leftarrow 0$ |
| SNAC | $rB_7K_0$ | $SC \leftarrow 0$ |
| SZAC | $rB_6K_7$ | $SC \leftarrow 0$ |
| PUSH | $D_7I'\ T_5\,B_6K_6$ | $SC \leftarrow 0$ |
| POP | $D_7I'\ T_4\,B_7K_5$ | $SC \leftarrow 0$ |
| SUB | $D_7I'\ T_4\,B_4K_4$ | $SC \leftarrow 0$ |
| ADC | $D_7I'\ T_4\,B_4K_3$ | $SC \leftarrow 0$ |
| XOR | $D_7I'\ T_4\,B_5K_0$ | $SC \leftarrow 0$ |
| OR | $D_7I'\ T_4\,B_4K_7$ | $SC \leftarrow 0$ |
| AND | $D_7I'\ T_4\,B_4K_6$ | $SC \leftarrow 0$ |
| SBB | $D_7I'\ T_4\,B_4K_5$ | $SC \leftarrow 0$ |

# The ALU and the accumulator

| CLAC | $rB_5K_3$ | $AC \leftarrow 0$ |
|------|-----------|-------------------|
| LDABP | $D_7I' T_3 B_6K_2$ | $AC \leftarrow 0$ |
| POP | $D_7I' T_3 B_6K_5$ | $AC \leftarrow 0$ |
| LDA | $D_2T_4$ | $AC \leftarrow 0$ |
| LDABP | $D_7I' T_5 B_6K_2$ | $AC \leftarrow AC + DR$ |
| POP | $D_7I' T_4 B_7K_5$ | $AC \leftarrow AC + DR$ |
| ADD | $D_7I' T_4 B_4K_2$ | $AC \leftarrow AC + DR$ |
| LDA | $D_2T_5$ | $AC \leftarrow AC + DR$ |
| INC | $rB_4K_0$ | $AC \leftarrow AC + 1$ |
| DEC | $rB_4K_1$ | $AC \leftarrow AC - 1$ |
| ADC | $D_7I' T_4 B_4K_3$ | $AC \leftarrow AC + DR + C$ |
| SBB | $D_7I' T_4 B_4K_5$ | $AC \leftarrow AC - DR + C$ |
| SUB | $D_7I' T_4 B_4K_4$ | $AC \leftarrow AC - DR$ |
| AND | $D_7I' T_4 B_4K_6$ | $AC \leftarrow AC \wedge DR$ |
| OR | $D_7I' T_4 B_4K_7$ | $AC \leftarrow AC \vee DR$ |
| XOR | $D_7I' T_4 B_5K_0$ | $AC \leftarrow AC \oplus DR$ |
| SHR | $rB_5K_1$ | $AC \leftarrow shr\ AC, AC(15) \leftarrow C, C \leftarrow AC(0)$ |
| SHL | $rB_5K_2$ | $AC \leftarrow shl\ AC, AC(0) \leftarrow C, C \leftarrow AC(15)$ |
| CMAC | $rB_5K_4$ | $AC \leftarrow AC'$ |

We have not seen this.
However before we go on to interpret this, we need to look at the design of the ALU.

# Summary so far

- Derive the control logic in RTL format
  - To allow us to move data around on the local bus that we have created in according to the step needed to execute an instruction.
  - Produce the signal to select the ALU at the appropriate time
    - Signal to start the arithmetic/logic operation
    - Get the operands into the correct location

- Next
  - Look into the design of the ALU
  - The connection of the signal to the ALU
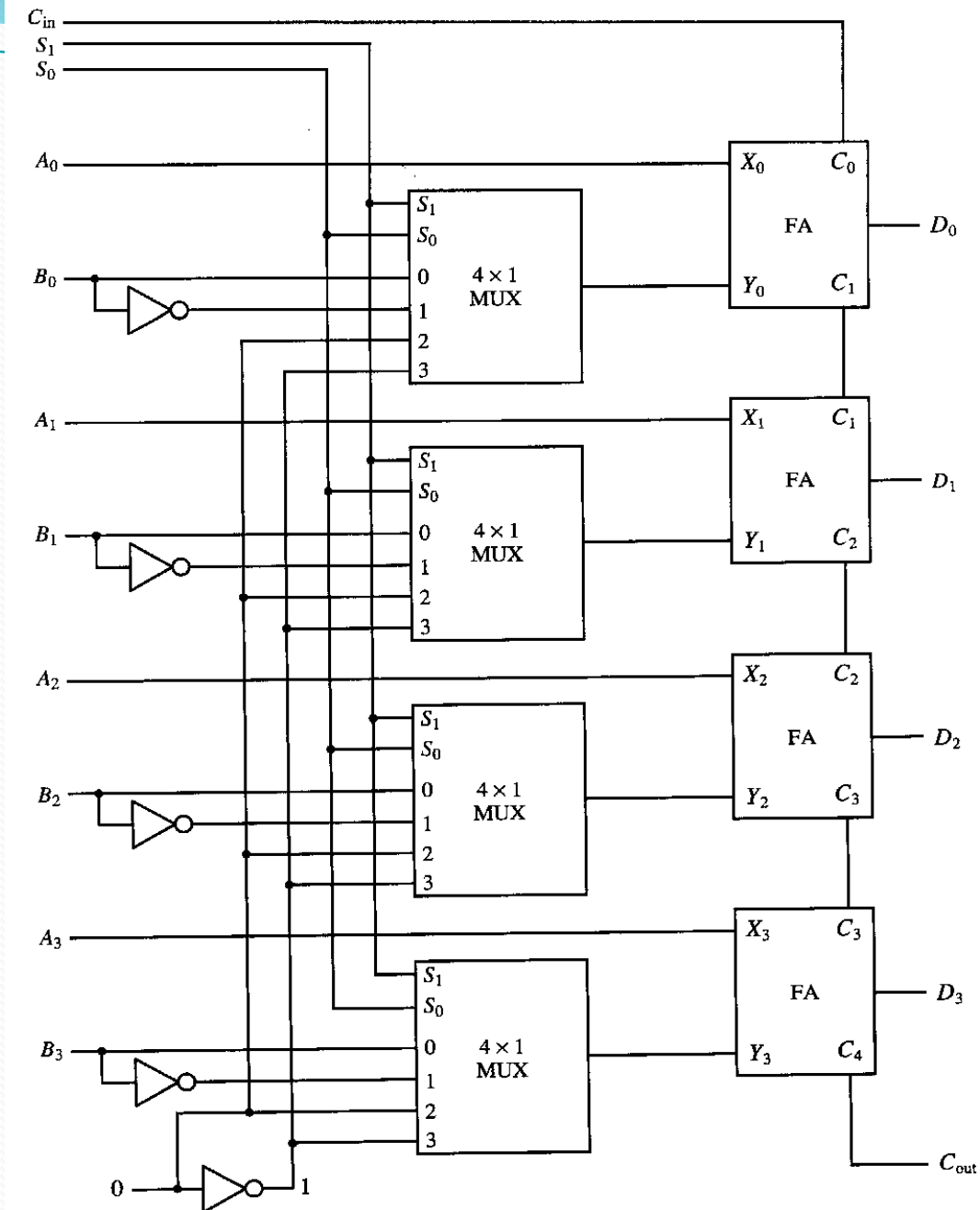
# Implementation of arithmetic operations
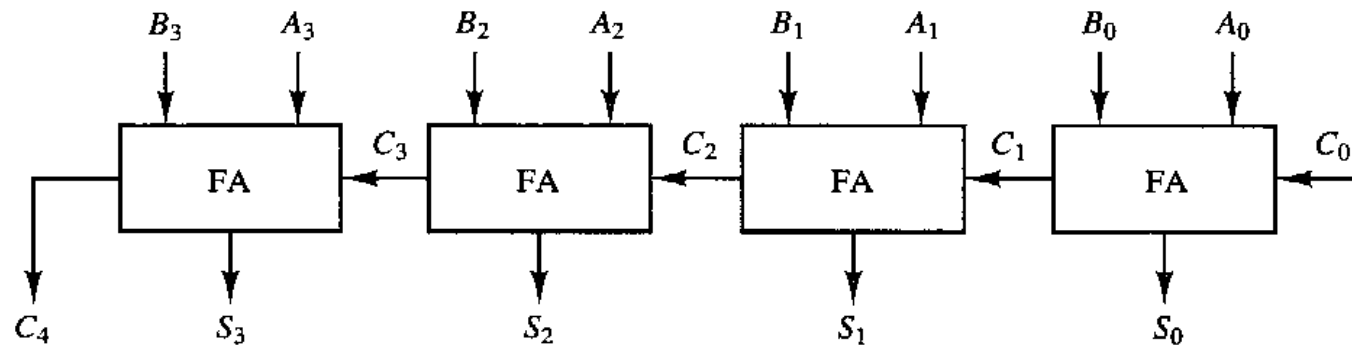


4-bit binary adder.
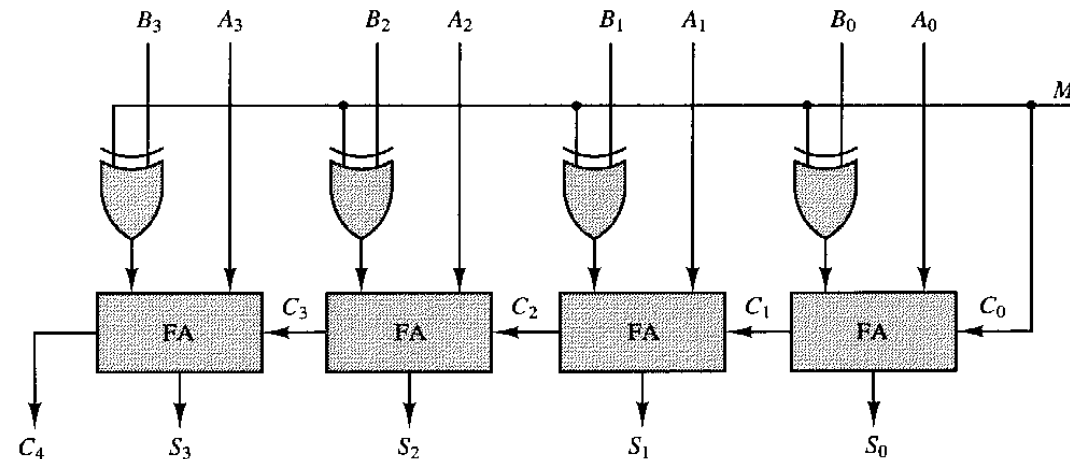


4-bit adder-subtractor.

# *Arithmetic micro-operations*

- A combination of the previous elementary circuits
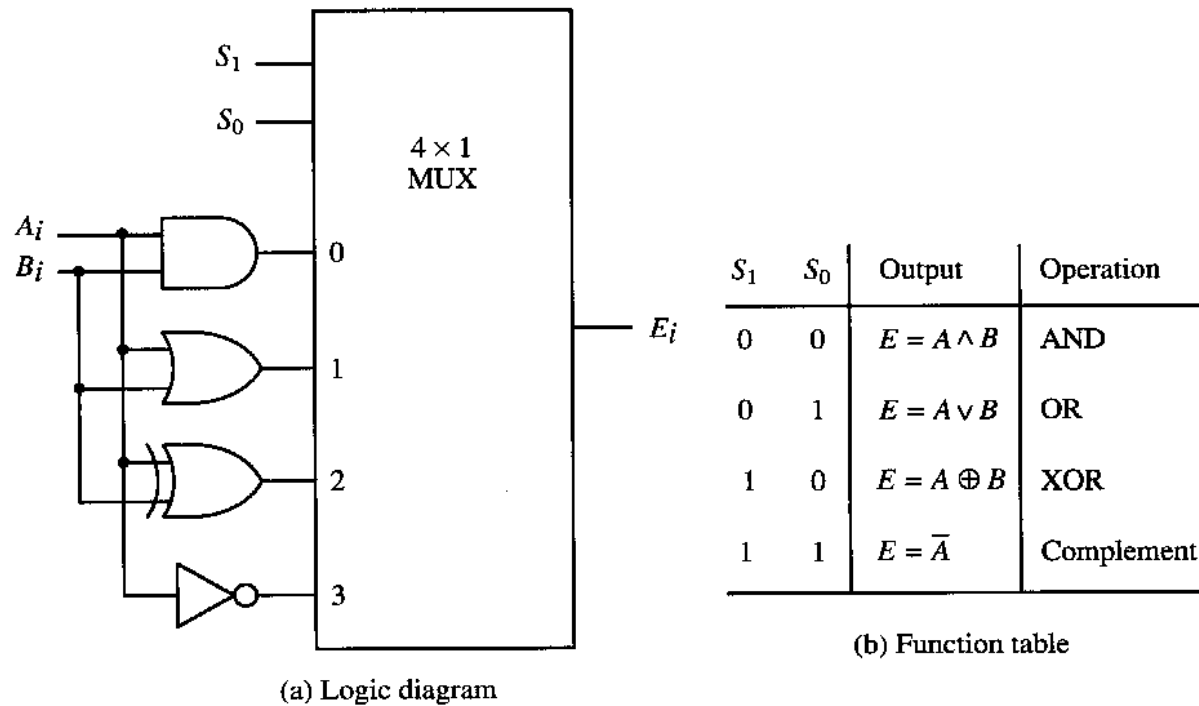  - add, adc, sub, sbb, inc, dec

# Arithmetic Circuit Function Table

| Select | | | Input | Output | Microoperation |
|---|---|---|---|---|---|
| $S_1$ | $S_0$ | $C_{in}$ | $Y$ | $D = A + Y + C_{in}$ | |
| 0 | 0 | 0 | $B$ | $D = A + B$ | Add |
| 0 | 0 | 1 | $B$ | $D = A + B + 1$ | Add with carry |
| 0 | 1 | 0 | $\overline{B}$ | $D = A + \overline{B}$ | Subtract with borrow |
| 0 | 1 | 1 | $\overline{B}$ | $D = A + \overline{B} + 1$ | Subtract |
| 1 | 0 | 0 | 0 | $D = A$ | Transfer $A$ |
| 1 | 0 | 1 | 0 | $D = A + 1$ | Increment $A$ |
| 1 | 1 | 0 | 1 | $D = A - 1$ | Decrement $A$ |
| 1 | 1 | 1 | 1 | $D = A$ | Transfer $A$ |

# Implementation of logic micro-operations

One stage of logic circuit



| $S_1$ | $S_0$ | Output | Operation |
|-------|-------|--------|-----------|
| 0 | 0 | $E = A \wedge B$ | AND |
| 0 | 1 | $E = A \vee B$ | OR |
| 1 | 0 | $E = A \oplus B$ | XOR |
| 1 | 1 | $E = \overline{A}$ | Complement |

(b) Function table

(a) Logic diagram

# Shift micro-operations

## Shift Microoperations

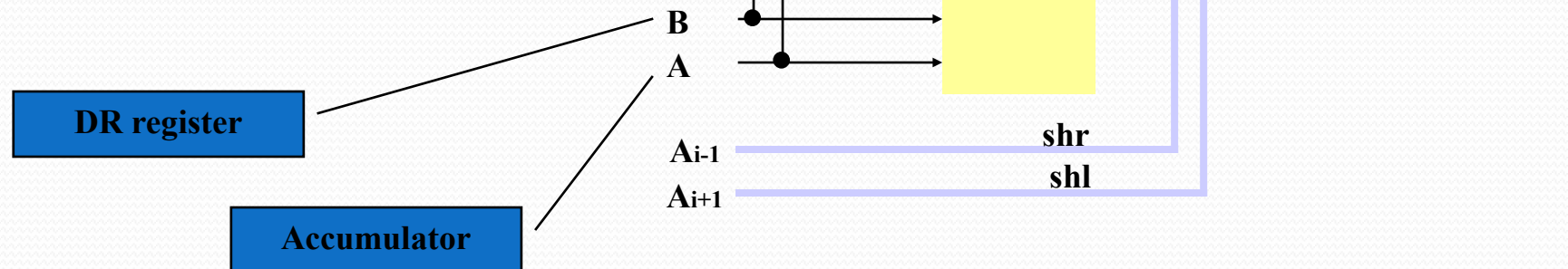| Symbolic designation | Description |
|---|---|
| $R \leftarrow \text{shl } R$ | Shift-left register $R$ |
| $R \leftarrow \text{shr } R$ | Shift-right register $R$ |
| $R \leftarrow \text{cil } R$ | Circular shift-left register $R$ |
| $R \leftarrow \text{cir } R$ | Circular shift-right register $R$ |
| $R \leftarrow \text{ashl } R$ | Arithmetic shift-left $R$ |
| $R \leftarrow \text{ashr } R$ | Arithmetic shift-right $R$ |



Arithmetic shift right.

Select
0 for shift right (down)
1 for shift left (up)

Serial input ($I_R$)

$A_0$
$A_1$
$A_2$
$A_3$

Serial input ($I_L$)

### Function table

| Select | Output | | | |
|---|---|---|---|---|
| $S$ | $H_0$ | $H_1$ | $H_2$ | $H_3$ |
| 0 | $I_R$ | $A_0$ | $A_1$ | $A_2$ |
| 1 | $A_1$ | $A_2$ | $A_3$ | $I_L$ |
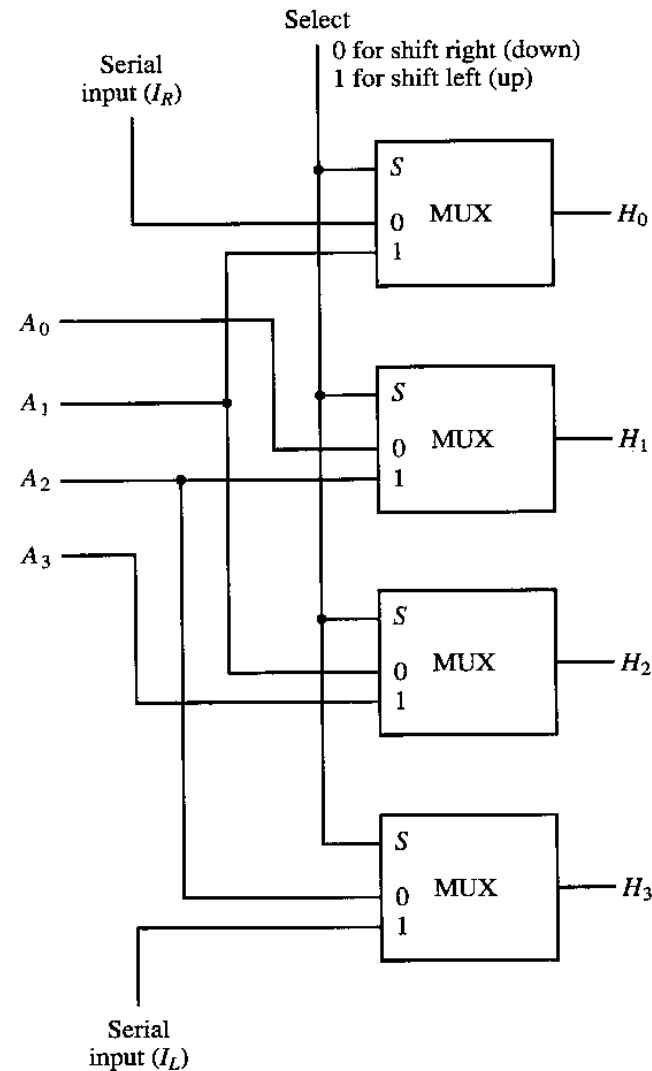
4-bit combinational circuit shifter.

# Arithmetic logic shift micro-operations

| S3 | S2 | S1 | S0 | Operation | Function |
|----|----|----|----|-----------|----------|
| 0 | 0 | 0 | 0 | F=A+1 | INC |
| 0 | 0 | 0 | 1 | F=A+B+Cin | ADD |
| 0 | 0 | 1 | 0 | F=A-B+Cin | SUB |
| 0 | 0 | 1 | 1 | F=A-1 | DEC |
| 0 | 1 | 0 | 0 | F=A∧B | AND |
| 0 | 1 | 0 | 1 | F=A∨B | OR |
| 0 | 1 | 1 | 0 | F=A⊕B | XOR |
| 0 | 1 | 1 | 1 | F=A' | COM |
| 1 | 0 | x | x | F=shr A | SHR |
| 1 | 1 | x | x | F=shl A | SHL |

# The ALU and the accumulator

| CLAC | $rB_5K_3$ | $AC \leftarrow 0$ |
|------|-----------|-------------------|
| LDABP | $D_7I' T_3 B_6 K_2$ | $AC \leftarrow 0$ |
| POP | $D_7I' T_3 B_6 K_5$ | $AC \leftarrow 0$ |
| LDA | $D_2 T_4$ | $AC \leftarrow 0$ |
| INC | $rB_4K_0$ | $AC \leftarrow AC + 1$ |
| ADC | $D_7I' T_4 B_4 K_3$ | $AC \leftarrow AC + DR + C$ |
| LDABP | $D_7I' T_5 B_6 K_2$ | $AC \leftarrow AC + DR + 0$ |
| POP | $D_7I' T_4 B_7 K_5$ | $AC \leftarrow AC + DR + 0$ |
| ADD | $D_7I' T_4 B_4 K_2$ | $AC \leftarrow AC + DR + 0$ |
| LDA | $D_2 T_5$ | $AC \leftarrow AC + DR + 0$ |
| SBB | $D_7I' T_4 B_4 K_5$ | $AC \leftarrow AC - DR + C$ |
| SUB | $D_7I' T_4 B_4 K_4$ | $AC \leftarrow AC - DR + 0$ |
| DEC | $rB_4K_1$ | $AC \leftarrow AC - 1$ |
| AND | $D_7I' T_4 B_4 K_6$ | $AC \leftarrow AC \wedge DR$ |
| OR | $D_7I' T_4 B_4 K_7$ | $AC \leftarrow AC \vee DR$ |
| XOR | $D_7I' T_4 B_5 K_0$ | $AC \leftarrow AC \oplus DR$ |
| CMAC | $rB_5K_4$ | $AC \leftarrow AC'$ |
| SHR | $rB_5K_1$ | $AC \leftarrow shr\ AC,$ $AC(15) \leftarrow C, C \leftarrow AC(0)$ |
| SHL | $rB_5K_2$ | $AC \leftarrow shl\ AC,$ $AC(0) \leftarrow C, C \leftarrow AC(15)$ |

$AC_{clr} = OR( \dots )$

N0

$N1 = OR( \dots )$

$N2 = OR( \dots )$

N3

N4

N5

N6

N7

N8

N12

**Decoding Table**

| S3 | S2 | S1 | S0 | |
|----|----|----|----|-----|
| 0 | 0 | 0 | 0 | N0 |
| 0 | 0 | 0 | 1 | N1 |
| 0 | 0 | 1 | 0 | N2 |
| 0 | 0 | 1 | 1 | N3 |
| 0 | 1 | 0 | 0 | N4 |
| 0 | 1 | 0 | 1 | N5 |
| 0 | 1 | 1 | 0 | N6 |
| 0 | 1 | 1 | 1 | N7 |
| 1 | 0 | x | x | N8 |
| 1 | 1 | x | x | N12 |

# ALU and Accumulator logic



$M = D7I'\ T4\ B4K3 \lor D7I'\ T4\ B4K5$

16 to 4 MUX, N0…N12, S3, S2, S1, S0

2-1 MUX, 0, C carry flag

$C_{in}$

Arithmetic circuit

$C_{out}$

$D_i$

Logic circuit

$E_i$

Sel

4x1 MUX, 0, 1, 2, 3, F

Accumulator

Clr    Ld=OR(N0….N12)

DR

AC

$AC_{i-1},\ (AC_{-1}=C)$

$AC_{i+1},\ (AC_{16}=C)$

shr

shl
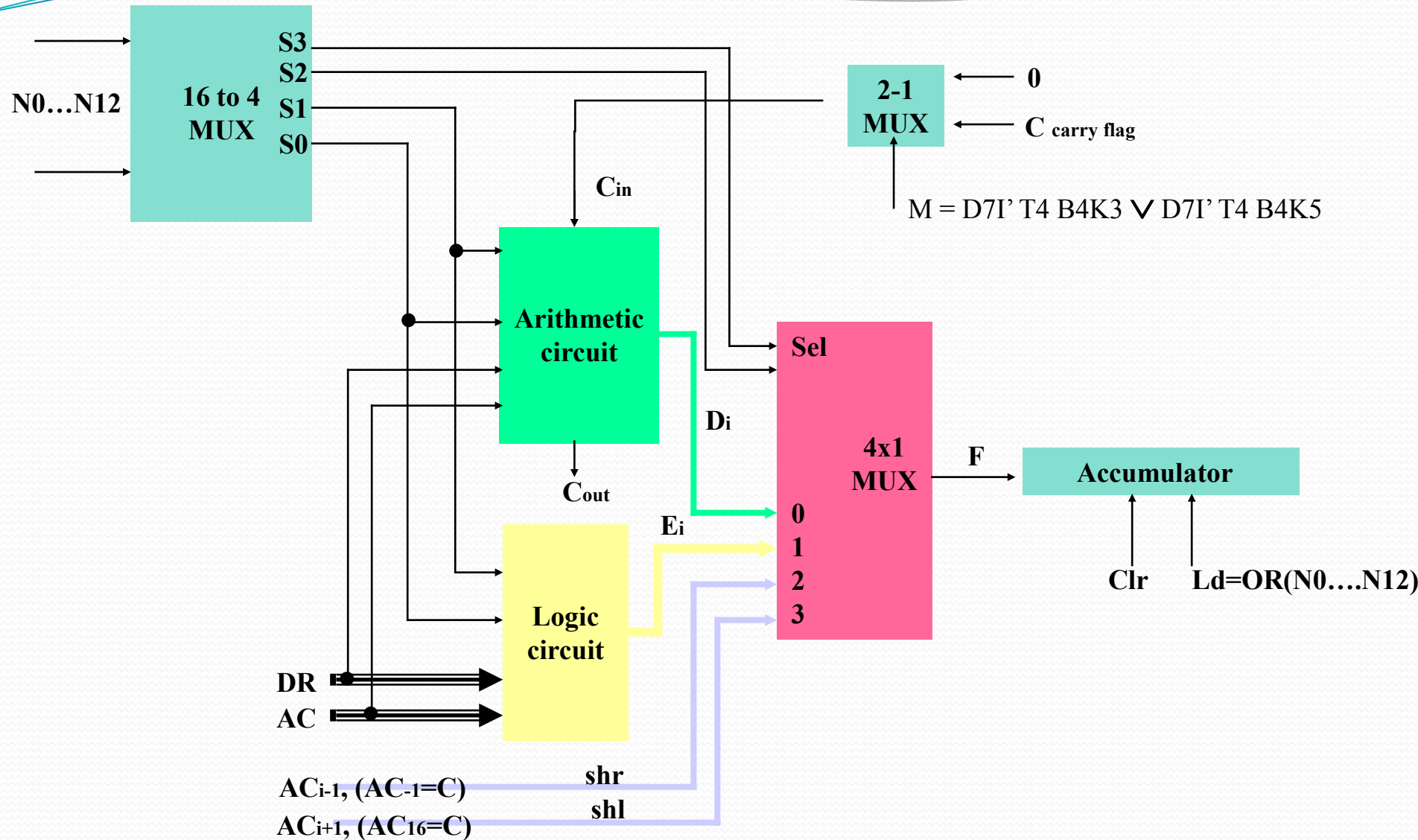
| Condition | Qk+1 | J | K |
|---|---|---|---|
| rB5K5 | 0 | 0 | 1 |
| rB8K6 | Qk' | 1 | 1 |
| rB5K7 | 1 | 1 | 0 |
| (N0∨N1∨N2∨N3) ∧ Cout | 1 | 1 | 0 |
| (N0∨N1∨N2∨N3) ∧ Cout' | 0 | 0 | 1 |
| N8 ∧ AC(0) | 1 | 1 | 0 |
| N8 ∧ AC(0)' | 0 | 0 | 1 |
| N9 ∧ AC(15) | 1 | 1 | 0 |
| N9 ∧ AC(15)' | 0 | 0 | 1 |
| NOR( above conditions) | Qk | 0 | 0 |

C (carry )

J    Q

Clock

K    $\overline{Q}$
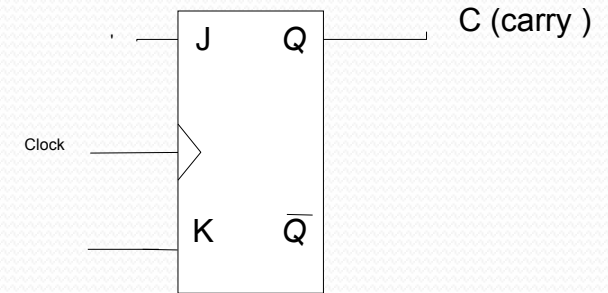
**The logic for J and K input can now be derive using a sum of product or a product of sum method.**

You have the logical design. Now it is a matter of coding them into VHDL/Verilog and then go through the placement and routing process.

# General Specification of CPUs

# General Specification of CPUs

- General register organization
- Stack organization
- Instruction formats
- Addressing modes
- Data transfer and manipulation
- Program control
- Reduced instruction set computer
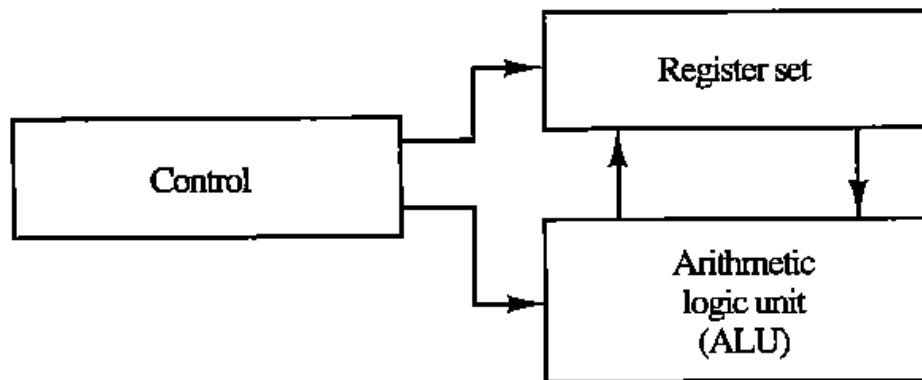


**Major components of CPU**

# General register organization

- Registers for intermediate results
  - bus interconnection
  - eg: R1 ← R2 + R3

1. MUX A selector (SELA): to place the content of R2 into bus A.
2. MUX B selector (SELB): to place the content of R3 into bus B.
3. ALU operation selector (OPR): to provide the arithmetic addition A + B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R1.
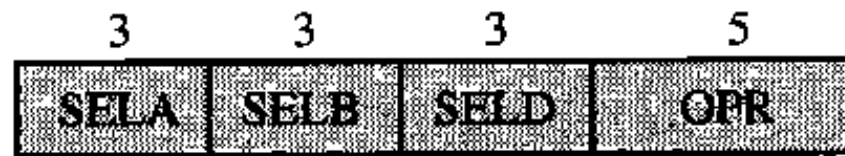
- Activated by control word



| 3 | 3 | 3 | 5 |
|------|------|------|-----|
| SELA | SELB | SELD | OPR |
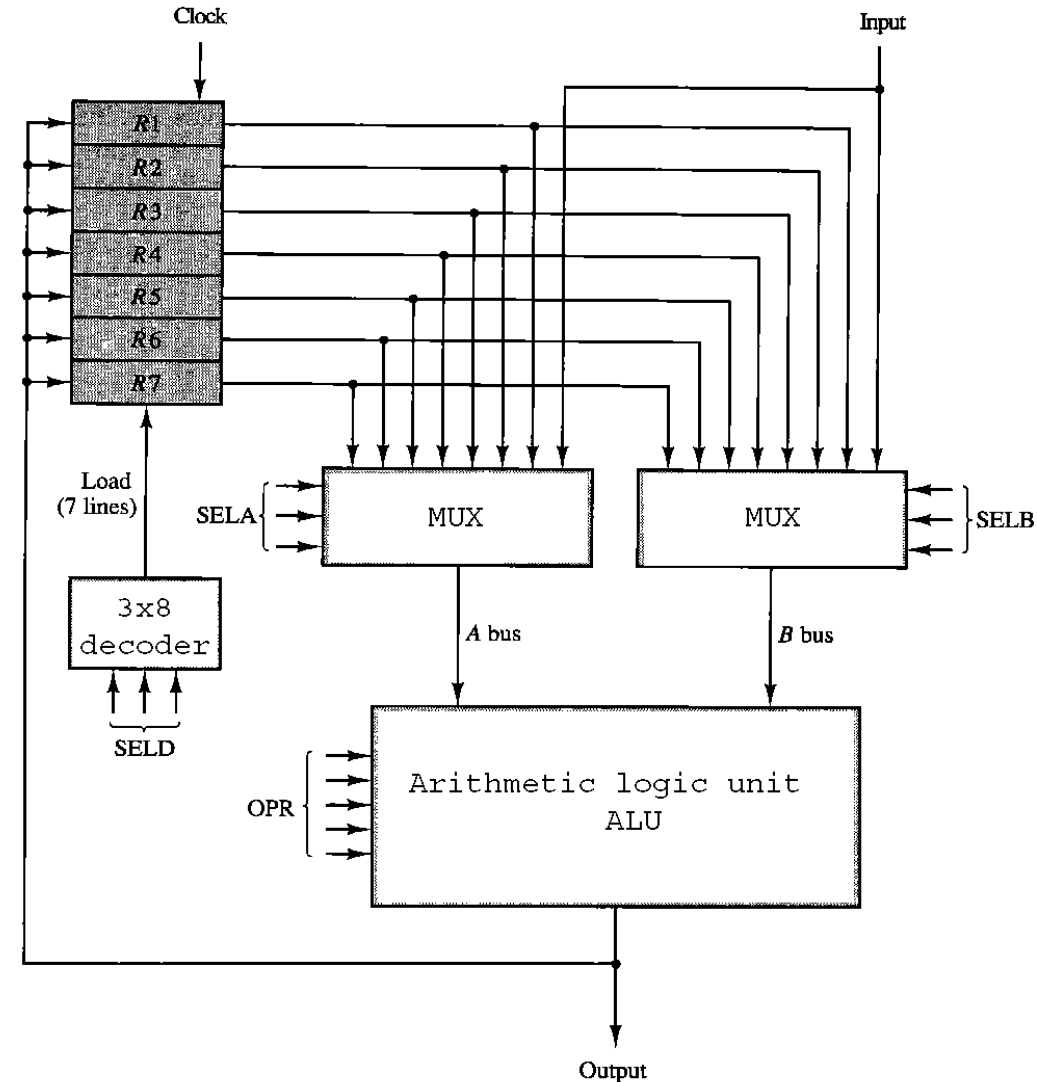
(a) Block diagram

(b) Control word

# Control word

| 3 | 3 | 3 | 5 |
|---|---|---|---|
| SELA | SELB | SELD | OPR |

## Encoding of ALU operations

## Encoding of register selection fields

| Binary Code | SELA | SELB | SELD |
|---|---|---|---|
| 000 | Input | Input | None |
| 001 | R1 | R1 | R1 |
| 010 | R2 | R2 | R2 |
| 011 | R3 | R3 | R3 |
| 100 | R4 | R4 | R4 |
| 101 | R5 | R5 | R5 |
| 110 | R6 | R6 | R6 |
| 111 | R7 | R7 | R7 |

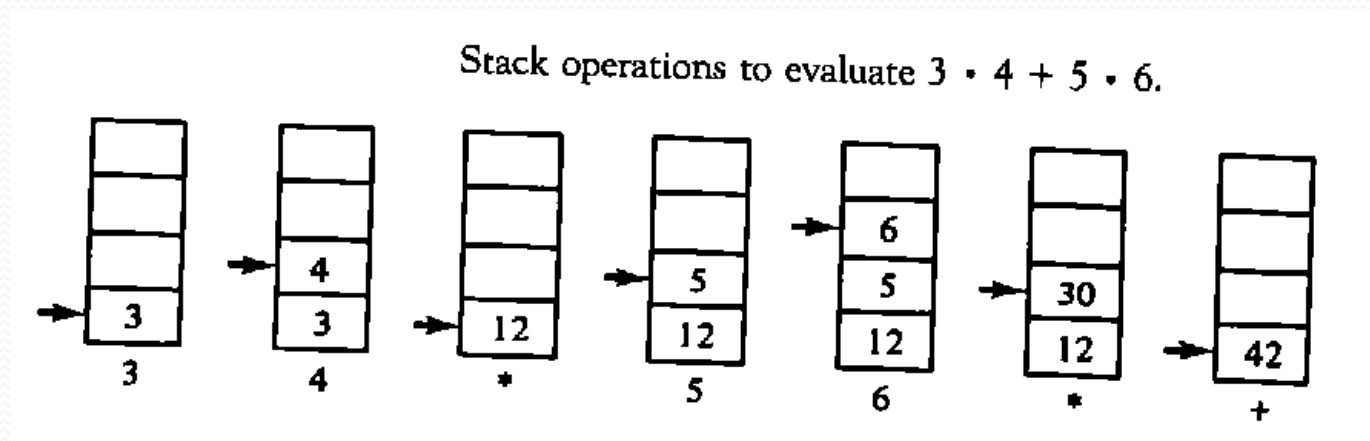| OPR Select | Operation | Symbol |
|---|---|---|
| 00000 | Transfer $A$ | TSFA |
| 00001 | Increment $A$ | INCA |
| 00010 | Add $A + B$ | ADD |
| 00101 | Subtract $A - B$ | SUB |
| 00110 | Decrement $A$ | DECA |
| 01000 | AND $A$ and $B$ | AND |
| 01010 | OR $A$ and $B$ | OR |
| 01100 | XOR $A$ and $B$ | XOR |
| 01110 | Complement $A$ | COMA |
| 10000 | Shift right $A$ | SHRA |
| 11000 | Shift left $A$ | SHLA |

# Stack Organization

- LIFO
- Typical for temporary storage of variables/parameters
- 2 operations
  - push (for registers)
    - SP←SP+1
    - M[SP] ←DR
    - if (SP=0) then (FULL ←1)
    - EMTY ←0
  - pop (for registers)
    - DR ← M[SP]
    - SP←SP-1
    - if (SP=0) then (EMTY ←1)
    - FULL ←0
- type
  - registers
  - memory
    - normally do not provide checks for overflows
    - normally grows downwards

**64 word stack registers**
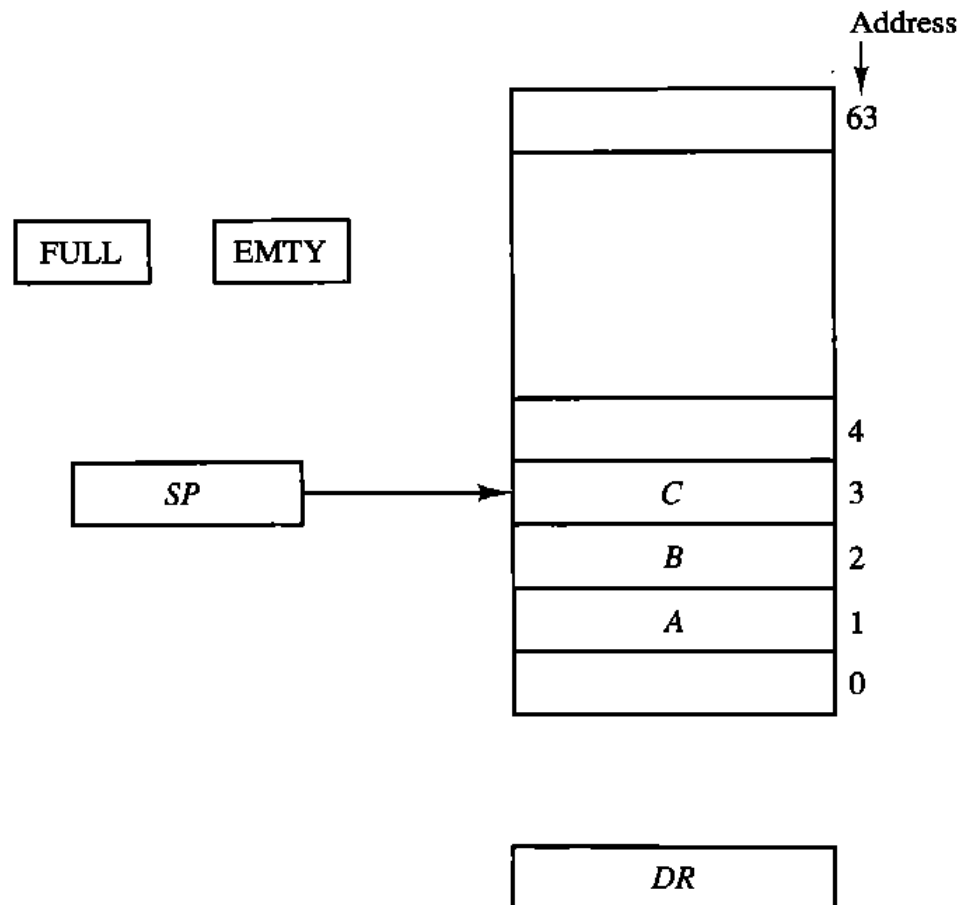
# Reverse polish notation

- Notation:
  - Infix:  A+B                    RPN:  AB+
  - infix: (A+B)*[C*(D+E)+F]       RPN:AB+DE+C*F+*
- Effective for arithmetic notation
  - any arithmetic expression can be expressed in parentheses-free polish notation
  - ideal for stack implementation

Stack operations to evaluate 3 · 4 + 5 · 6.

| | 3 | | | | | |
|---|---|---|---|---|---|---|
| | 4 | | 5 | 6 | | |
| 3 | 3 | 12 | 12 | 5 | 30 | |
| | | | | 12 | 12 | 42 |
| 3 | 4 | * | 5 | 6 | * | + |

# CPU organisation: 3 types

- Single accumulator organisation
  - all instruction performed with an implied accumulator
    - ADD      X
- General register organisation
  - will require in general 3 register specification
    - ADD R1, R2, R3,      R1 ← R2 + R3
  - or 2 register specification, if destination is same as one of the source
    - ADD  R2, R3,     R2 ← R2 + R3
- Stack organisation
  - as discussed before where the main instruction is the PUSH
  - no need for address field
    - ADD
    - the operands are pop off the stack, result are push back automatically on to the stack as well

# Instruction formats

X=(A+B)*(C+D)

**Three address instruction**

```
ADD     R1, A, B      R1←M[A] + M[B]
ADD     R2, C, D      R2←M[C] + M[D]
MUL     X, R1, R2     M[X]←R1*R2
```

**Two address instruction**

```
MOV     R1, A     R1←M[A]
ADD     R1, B     R1←R1 + M[B]
MOV     R2, C     R2←M[C]
ADD     R2, D     R2←R2 + M[D]
MUL     R1,R2     R1←R1*R2
MOV     X, R1     M[X]←R1
```

**One address instruction**

```
LOAD    A     AC←M[A]
ADD     B     AC←AC + M[B]
STORE   T     M[T]←AC
LOAD    C     AC←M[C]
ADD     D     AC←AC + M[D]
MUL     T     AC←AC*M[T]
STORE   X     M[X]←AC
```

**Zero address instruction**

```
PUSH    A     TOS←A
PUSH    B     TOS←B
ADD           TOS←(A + B)
PUSH    C     TOS←C
PUSH    D     TOS←D
ADD           TOS←(C + D)
MUL           TOS←(C + D)*(A + B)
POP     X     M[X]←TOS
```

# RISC instructions

- Load-store architecture:
  - Use of only load and store instructions only to access memory
- All other instructions are register type
- Simplifies decoding – allow less work to be done in a clock cycle
  - Leads to better potential for higher frequency operation.

*X=(A+B)\*(C+D)*

```
LOAD      R1, A           R1 ← M[A]
LOAD      R2, B           R2 ← M[B]
LOAD      R3, C           R3 ← M[C]
LOAD      R4, D           R4 ← M[D]
ADD       R1, R1, R2      R1 ← R1 + R2
ADD       R3, R3, R2      R3 ← R3 + R4
MUL       R1, R1, R3      R1 ← R1 * R3
STORE     X, R1           M[X] ← R1
```

# Addressing modes

- To give programming versatility
- to reduce the number of bits in the addressing field of the instruction
- The various modes
  - Implied mode
  - Immediate mode
  - Register mode
  - Register indirect mode
  - Autoincrement or autodecrement mode
  - Direct address mode
  - Indirect address mode
  - Relative address mode
  - Indexed addressing mode
  - Base register addressing mode

# Classification of instructions

- Data transfer instructions
- Data manipulation instructions
- Program control instructions

# Data transfer instructions

| Name | Mnemonic |
|------|----------|
| Load | LD |
| Store | ST |
| Move | MOV |
| Exchange | XCH |
| Input | IN |
| Output | OUT |
| Push | PUSH |
| Pop | POP |

**Typical data transfer instructions**

**Recommended assembly language convention**

| Mode | Assembly Convention | Register Transfer |
|------|---------------------|-------------------|
| Direct address | LD ADR | $AC \leftarrow M[ADR]$ |
| Indirect address | LD @ADR | $AC \leftarrow M[M[ADR]]$ |
| Relative address | LD $ADR | $AC \leftarrow M[PC + ADR]$ |
| Immediate operand | LD #NBR | $AC \leftarrow NBR$ |
| Index addressing | LD ADR(X) | $AC \leftarrow M[ADR + XR]$ |
| Register | LD R1 | $AC \leftarrow R1$ |
| Register indirect | LD (R1) | $AC \leftarrow M[R1]$ |
| Autoincrement | LD (R1)+ | $AC \leftarrow M[R1], R1 \leftarrow R1 + 1$ |

**Various addressing modes**

# Data manipulation instructions

**TABLE 8-7** Typical Arithmetic Instructions

| Name | Mnemonic |
|---|---|
| Increment | INC |
| Decrement | DEC |
| Add | ADD |
| Subtract | SUB |
| Multiply | MUL |
| Divide | DIV |
| Add with carry | ADDC |
| Subtract with borrow | SUBB |
| Negate (2's complement) | NEG |

**TABLE 8-8** Typical Logical and Bit Manipulation Instructions

| Name | Mnemonic |
|---|---|
| Clear | CLR |
| Complement | COM |
| AND | AND |
| OR | OR |
| Exclusive-OR | XOR |
| Clear carry | CLRC |
| Set carry | SETC |
| Complement carry | COMC |
| Enable interrupt | EI |
| Disable interrupt | DI |

**TABLE 8-9** Typical Shift Instructions

| Name | Mnemonic |
|---|---|
| Logical shift right | SHR |
| Logical shift left | SHL |
| Arithmetic shift right | SHRA |
| Arithmetic shift left | SHLA |
| Rotate right | ROR |
| Rotate left | ROL |
| Rotate right through carry | RORC |
| Rotate left through carry | ROLC |

# Program control

## Typical Program Control Instructions

| Name | Mnemonic |
|------|----------|
| Branch | BR |
| Jump | JMP |
| Skip | SKP |
| Call | CALL |
| Return | RET |
| Compare (by subtraction) | CMP |
| Test (by ANDing) | TST |
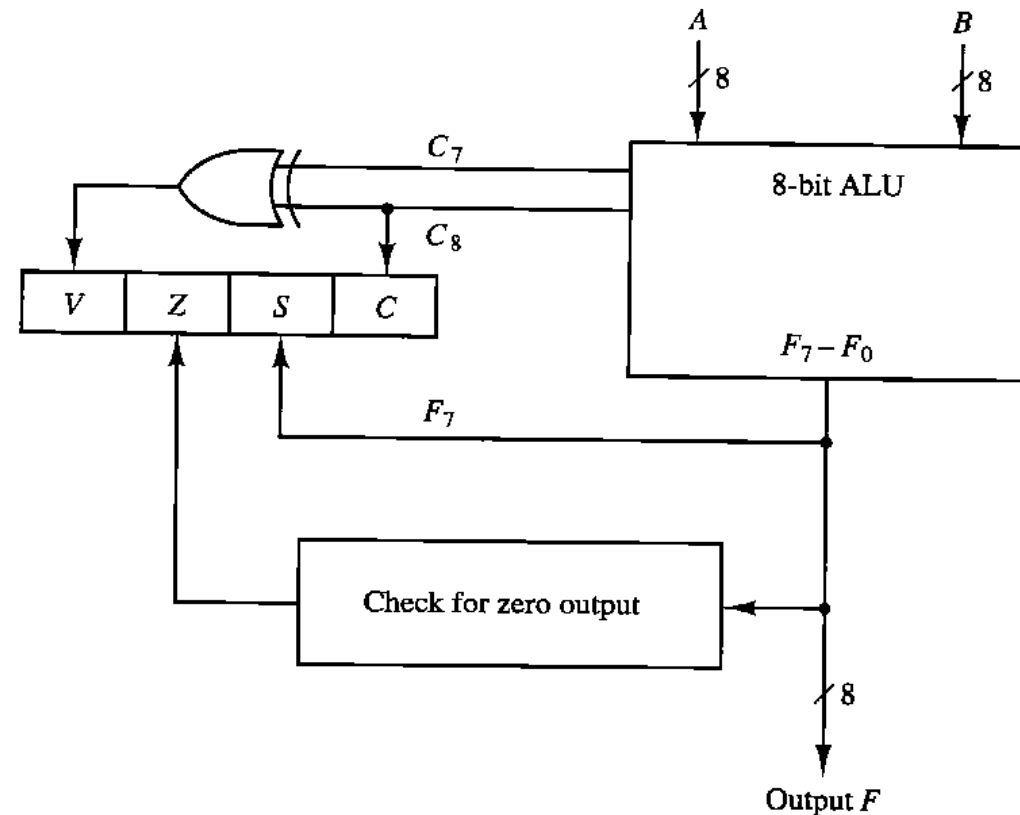


Status register bits.

$$
\begin{array}{rl}
A: & 11110000 \\
\bar{B} + 1: & +11101100 \\
\hline
A - B: & 11011100
\end{array}
$$

$$C = 1 \qquad S = 1 \qquad V = 0 \qquad Z = 0$$

# Conditional branch instructions

| Mnemonic | Branch condition | Tested condition |
|----------|------------------|------------------|
| BZ | Branch if zero | $Z = 1$ |
| BNZ | Branch if not zero | $Z = 0$ |
| BC | Branch if carry | $C = 1$ |
| BNC | Branch if no carry | $C = 0$ |
| BP | Branch if plus | $S = 0$ |
| BM | Branch if minus | $S = 1$ |
| BV | Branch if overflow | $V = 1$ |
| BNV | Branch if no overflow | $V = 0$ |
| *Unsigned* compare conditions $(A - B)$ | | |
| BHI | Branch if higher | $A > B$ |
| BHE | Branch if higher or equal | $A \geq B$ |
| BLO | Branch if lower | $A < B$ |
| BLOE | Branch if lower or equal | $A \leq B$ |
| BE | Branch if equal | $A = B$ |
| BNE | Branch if not equal | $A \neq B$ |
| *Signed* compare conditions $(A - B)$ | | |
| BGT | Branch if greater than | $A > B$ |
| BGE | Branch if greater or equal | $A \geq B$ |
| BLT | Branch if less than | $A < B$ |
| BLE | Branch if less or equal | $A \leq B$ |
| BE | Branch if equal | $A = B$ |
| BNE | Branch if not equal | $A \neq B$ |