ELE432

ADVANCED DIGITAL DESIGN

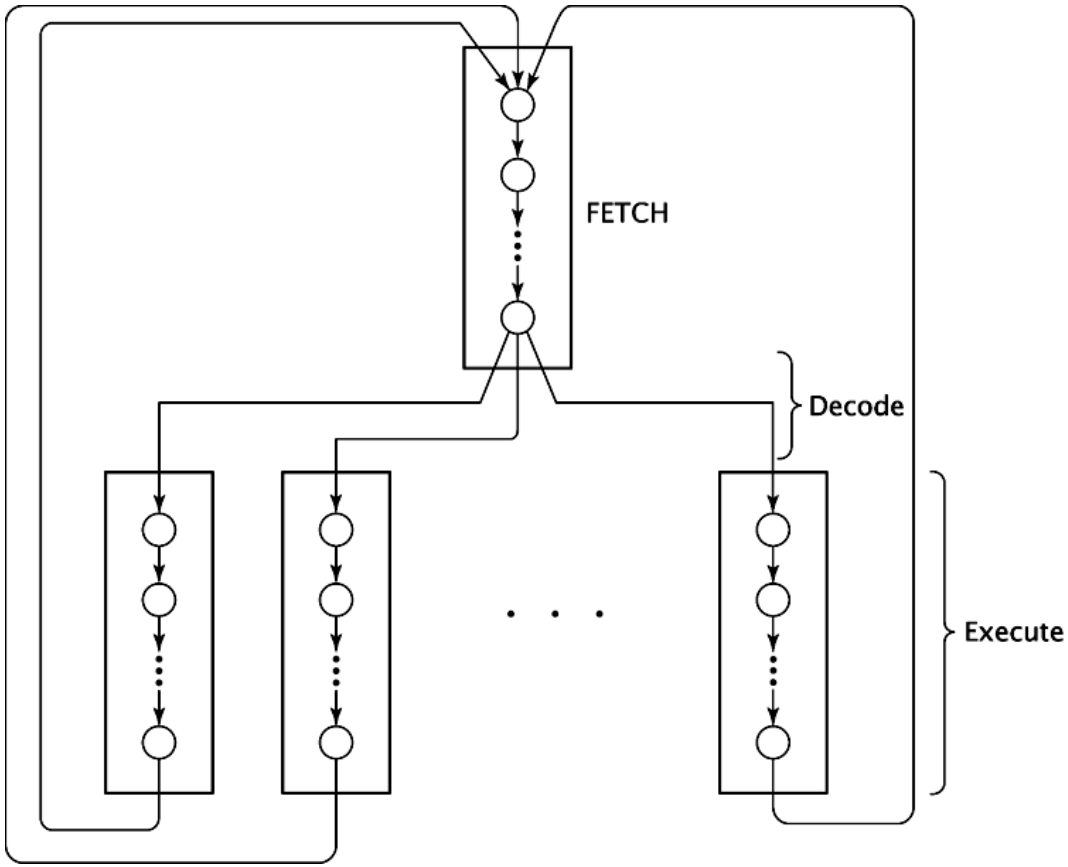
HACETTEPE UNIVERSITY

CPU DESIGN

Textbook: Computer Systems Organization & Architecture, John D. *Carpinelli*

Generic CPU State Diagram

- Fetch cycle: Fetch an instruction from memory, then go to the decode cycle.
- Decode cycle: Decode the instruction – that is, determine which instruction has been fetched – then go to the execute cycle for that instruction.
- Execute cycle; Execute the instruction, then go to the fetch cycle and fetch the next instruction.



Very Simple CPU

- Access 64 bytes of memory
- Each byte has 8 bits
- Six bit address onoutput pins A[5..0]
- Reads 8 bits from memory on inputs D[7..0]
- CPU has on one programmer-accessable register, an 8-bit accumulator, AC

Instruction	Instruction Code	Operation
ADD	00AAAAAA	AC ← AC + M[AAAAAA]
AND	01AAAAAA	AC ← AC ^ M[AAAAAA]
JMP	10AAAAAA	GOTO AAAAAA
INC	11XXXXXX	AC ← AC + 1

Very Simple CPU – Additional Registers

- Six bit address register, AR
- Six bit program counter, PC
- Eight bit data register, DR
- Two bit instruction register, IR

Very Simple CPU – Fetching Instructions from Memory

- Send the address to memory by placing it on the address pins A[5..0]
- After allowing memory enough time to perform its internal decoding and to retriwe ve the desired instruction, send a signal to memory so that it outputs the instruction on its output pins.
- The address is stored in the program counter, PC

  FETCH1: AR ← PC
- CPU asserts a READ signal and increments PC

  FETCH2: DR ← M, PC ← PC + 1
- Copy two high order bits of DR to IR and six low-order bits of DR to AR

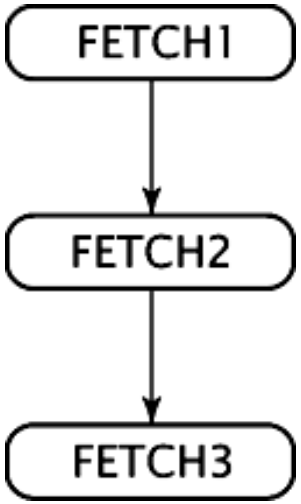
  FETCH3: IR ← DR[7..6], AR ← DR[5..0]

Fetch Cycle for Very Simple CPU

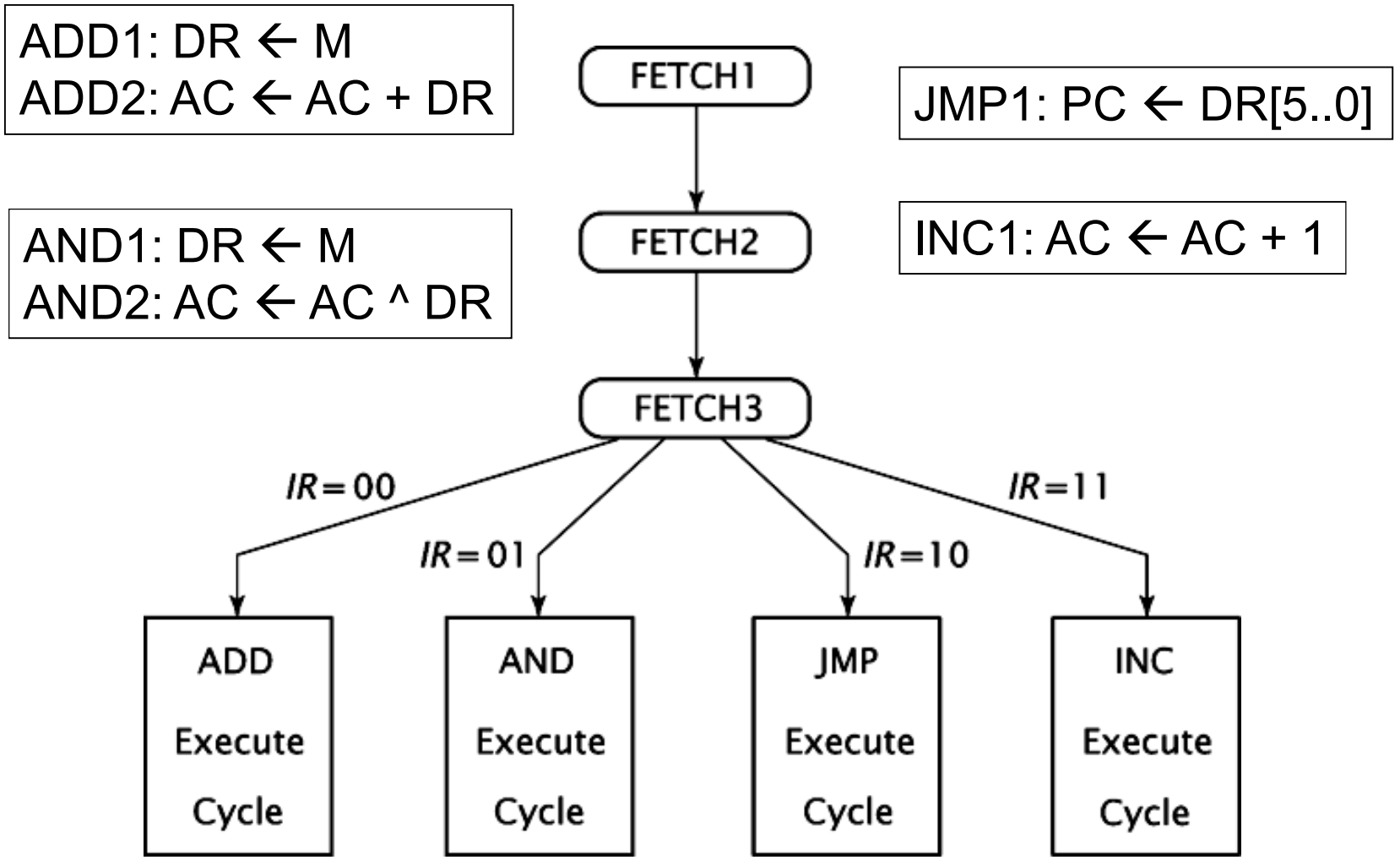
FETCH1: AR ← PC

FETCH2: DR ← M, PC ← PC + 1

FETCH3: IR ← DR[7..6], AR ← DR[5..0]

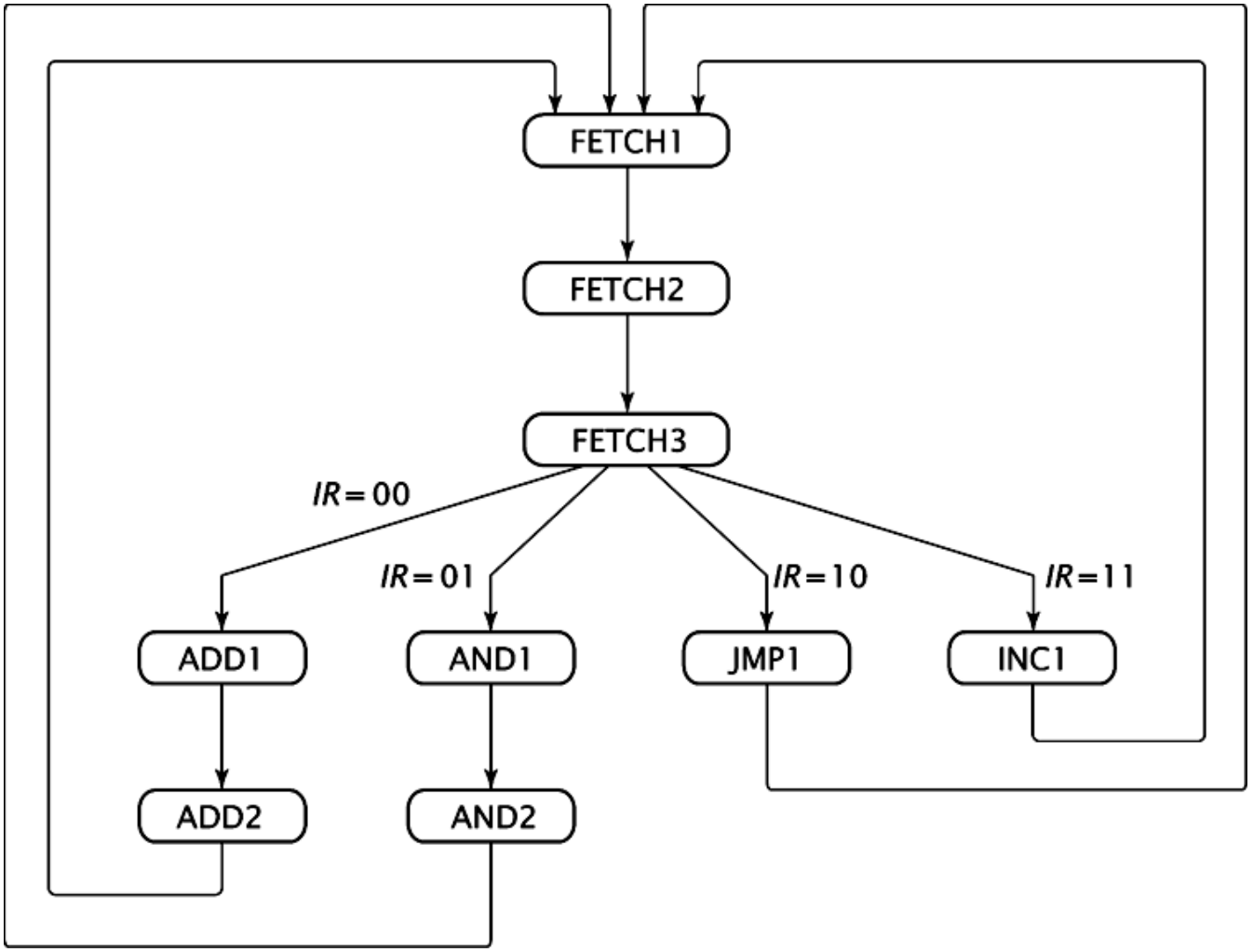


Fetch and Decode Cycles for Very Simple CPU



Complete State Diagram for Very Simple CPU

This diagram includes the data paths for all the operations



Establishing Required Data Paths

- Operations associated with each state of the CPU are:

  FETCH1: AR ← PC

  FETCH2: DR ← M, PC ← PC + 1

  FETCH3: IR ← DR[7..6], AR ← DR[5..0]

  ADD1: DR ← M

  ADD2: AC ← AC + DR

  AND1: DR ← M

  AND2: AC ← AC ^ DR

  JMP1: PC ← DR[5..0]

  INC1: AC ← AC + 1

Establishing Required Data Paths

- Regroup the operations, without regard for the cycles in which they occur, by registers whose contents they modify.

  AR: AR ← PC; AR ← DR[5..0]

  PC: PC ← PC + 1; PC ← DR[5..0]

  DR: DR ← M

  IR: IR ← DR[7..6]

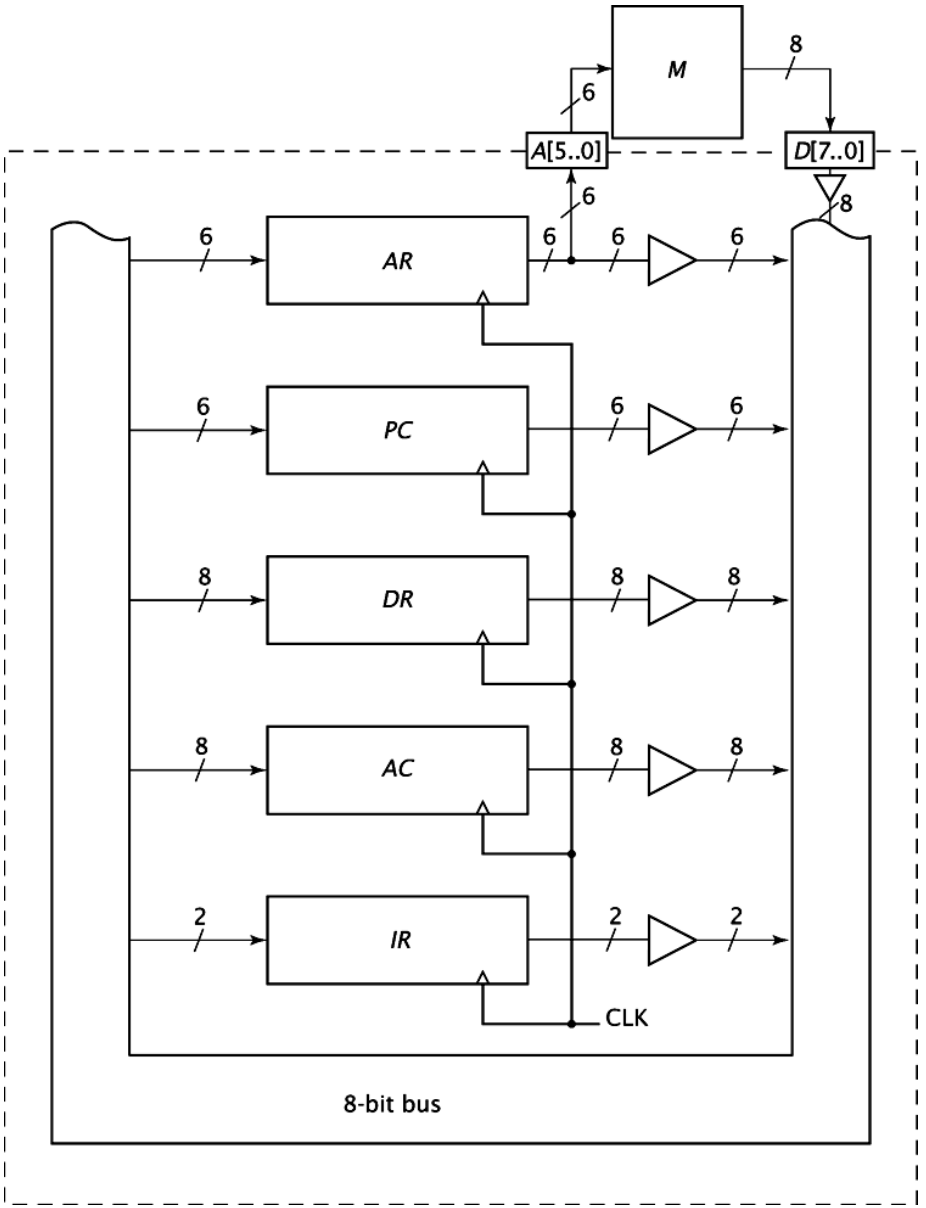
  AC: AC ← AC + DR; AC ← AC ^ DR; AC ← AC + 1

- Connect every component to the system bus

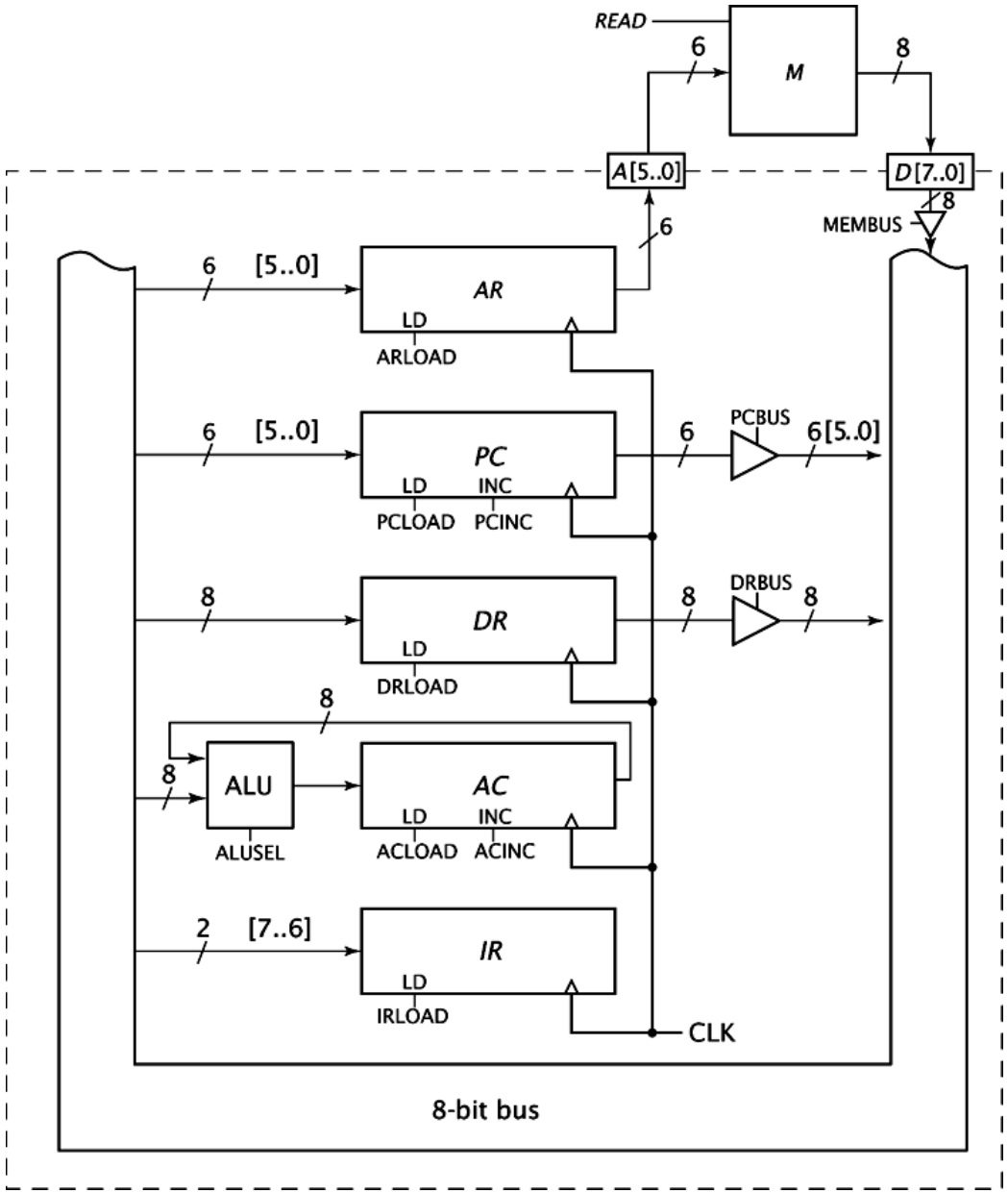
Preliminary Register Section for the Very Simple CPU

Simple Observations:

1. AR only supplies its data to memory – don't need to connect to any other component
2. IR does not supply data to any other component via the internal bus – can remove output connection
3. AC does not supply itws data to any component – remove connection to internal bus
4. The bus is 8 bits wide, but not all data transfers are 8 bits (some are 6, some are 2) – need to specify which registers send and receive to and from which bus
5. AC must be able to load the sum of AC and DR, logical AND of AC and DR. CPU needs to include an ALU to do that.

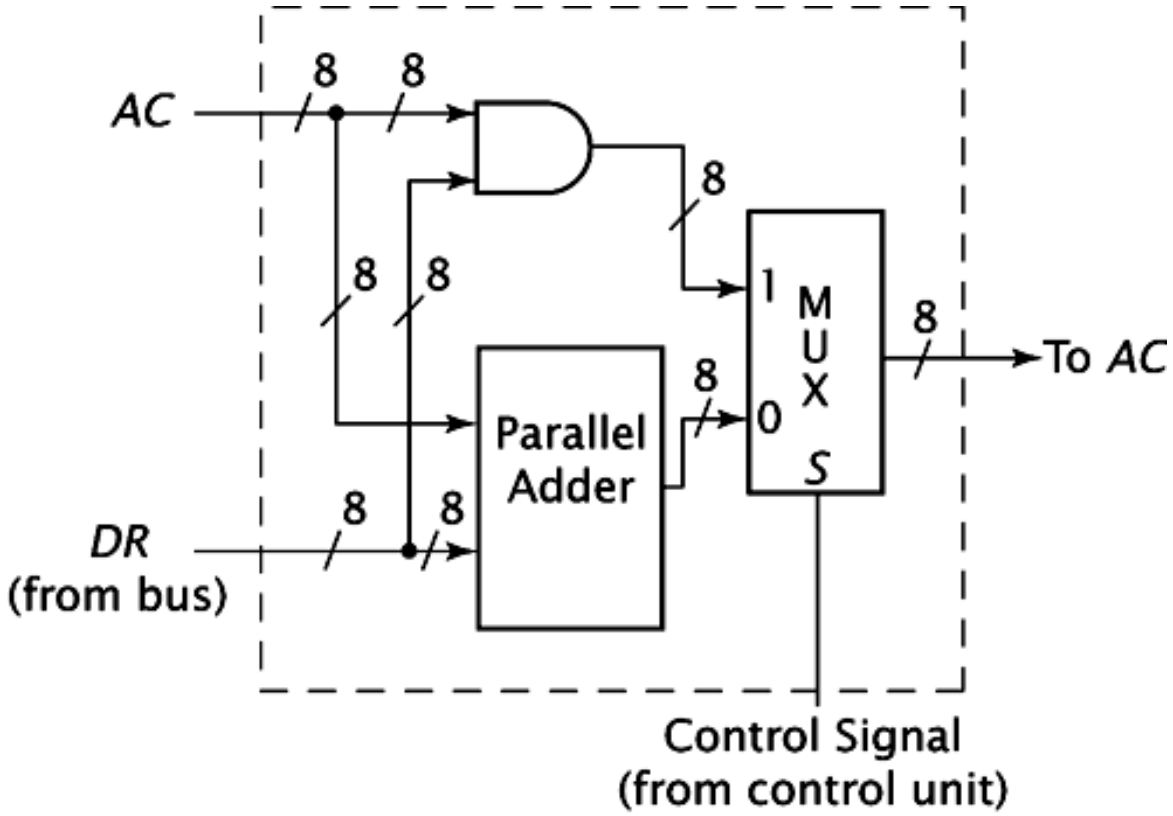


Final Register Section for the Very Simple CPU

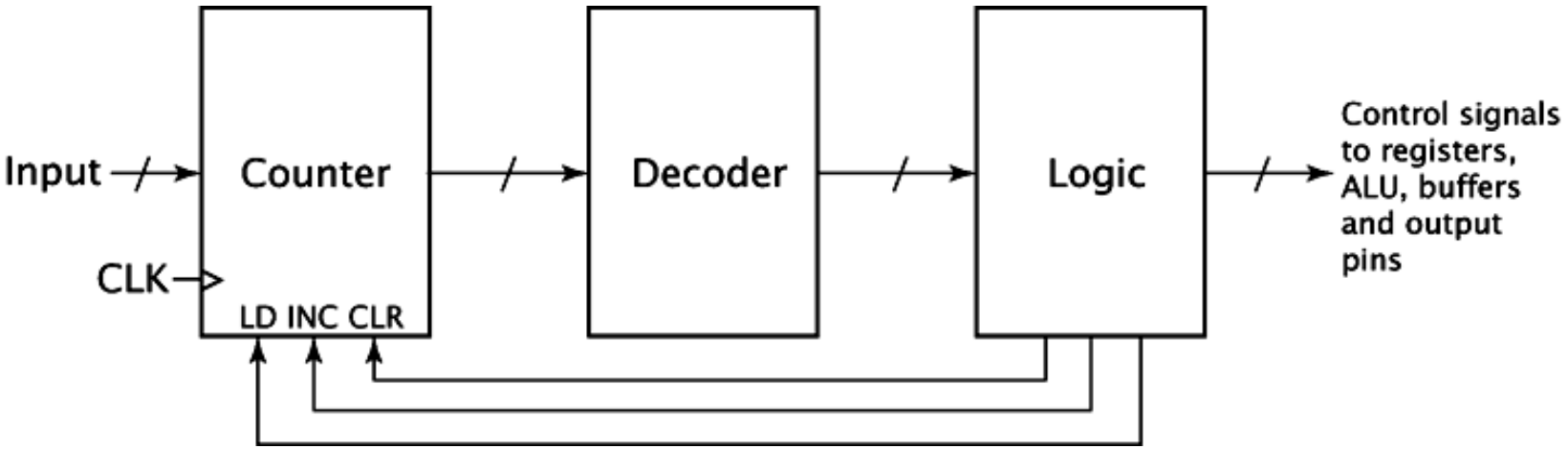


A Very Simple ALU

Create separate hardware for each function and use a multiplexer to select the function results



Generic Hardwired Control Unit



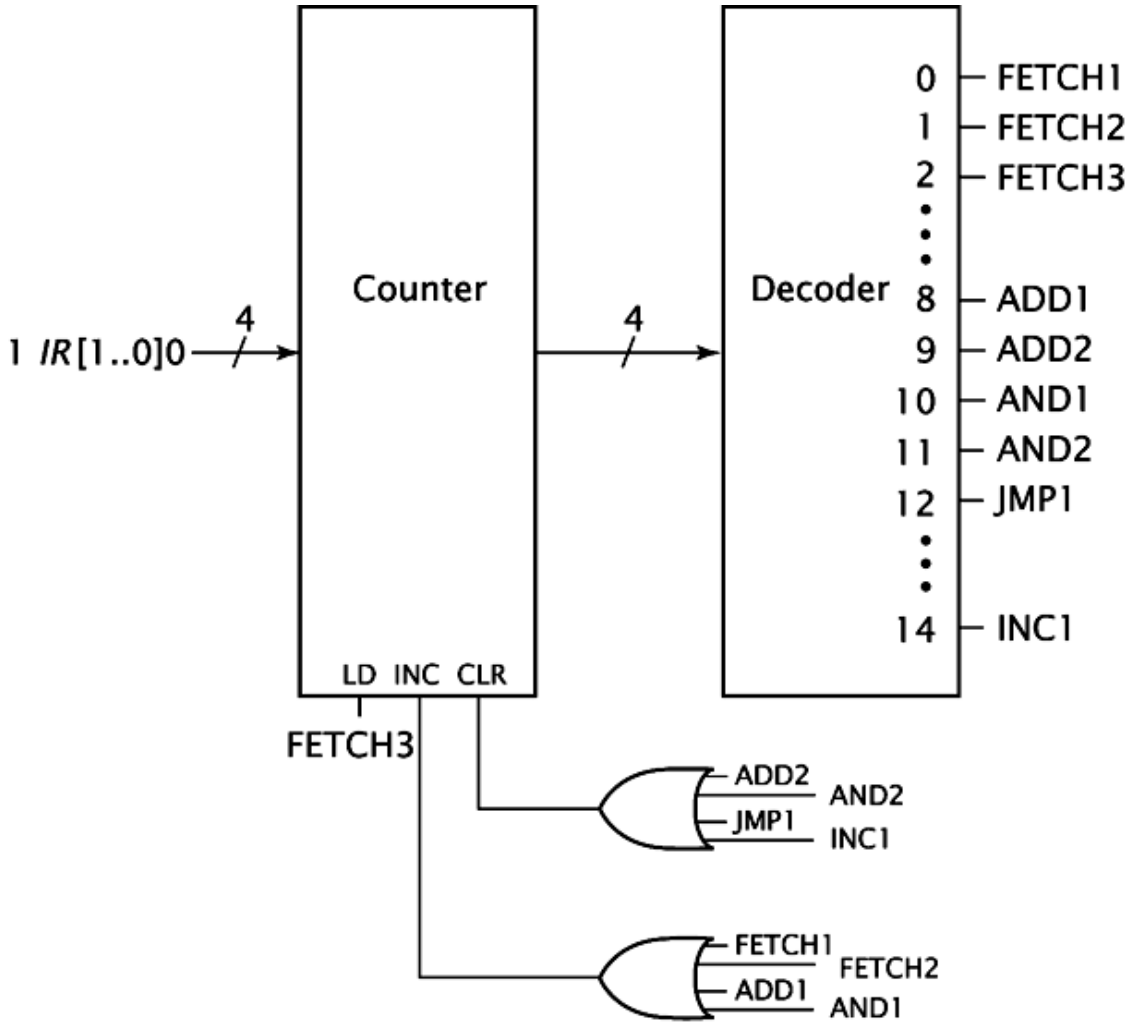
- Counter contains current state
- Generates individual signals from current state
- Logic to take individual state signals and generate control signals for each component, as well as the signals to control counter

Very Simple CPU has Nine States

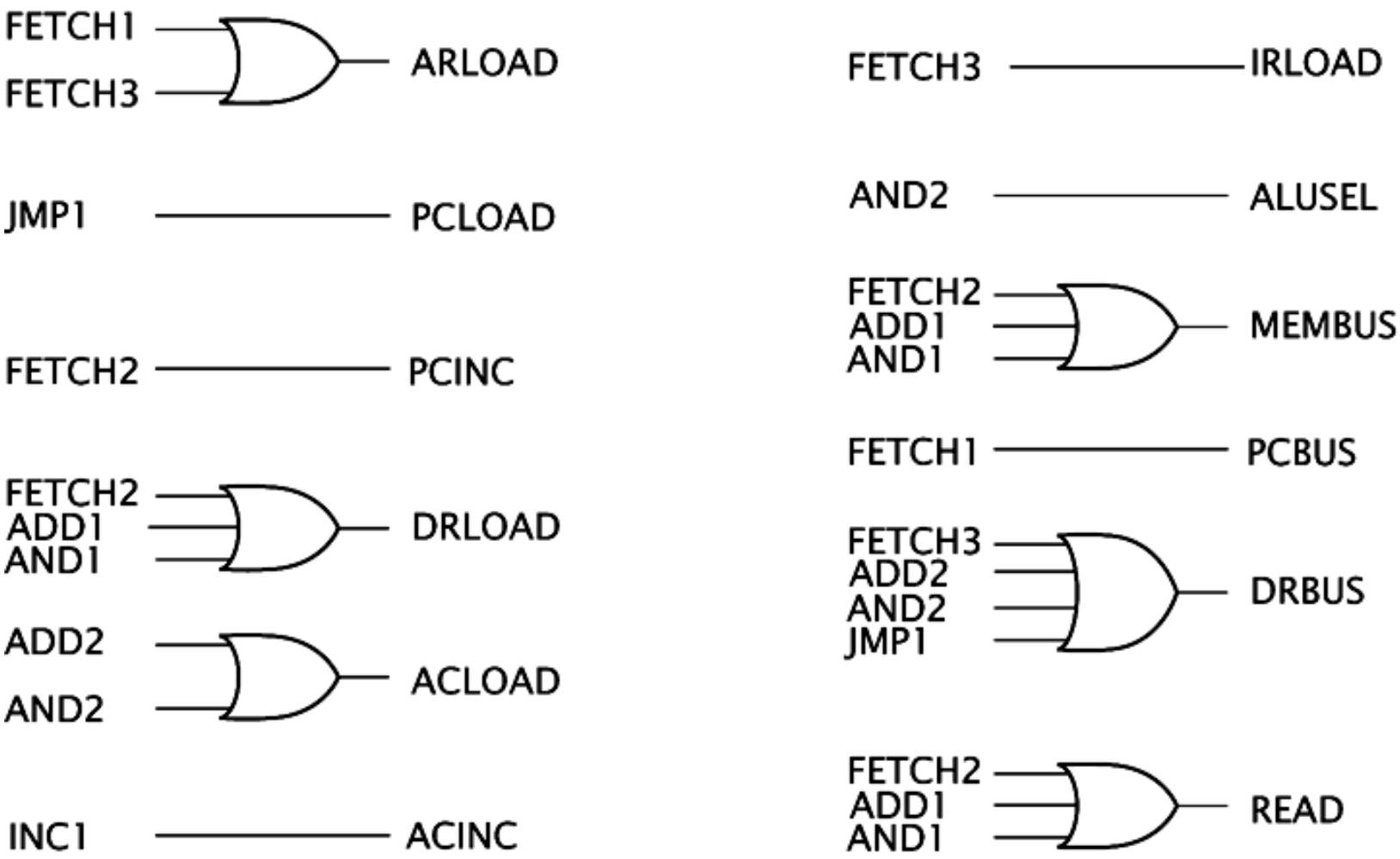
- Need four bit counter and a 4-to-16 bit decoder. Seven outputs of decoder will not be used

1. Assign FETCH1 to counter value 0 and use the CLR input of the counter to reach this state.
2. Assign sequential states to sequential counter values and use the INC input of the counter to traverse these states. CPU would assign FETCH2 to counter value 1 and FETCH3 to counter value 2, ADD1 and ADD2 to consecutive counter values as well as AND1 and AND2
3. Assign the first state fo each execute routine based on the instruction opcodes and the maximum number of states in the execute routines. Use the opcodes to generate the data input to the counter and the LD input to the counter to reach the proper execute routine.

Hardwired Control Unit for Very Simple CPU



Control Signal Generation for Very Simple CPU



Relatively Simple CPU

- Access 64K bytes of memory
- Each byte 8 bits wide
- Address pins A[15..0]
- Bidirectional data pins D[7..0]
- Programmer directly controls 3 registers
  - AC – 8 bit accumulator
  - R – 8 bit supplier of second operand
  - Z – one bit flag register set by arithmetic or logic operations

Instruction Set for a Relatively Simple CPU

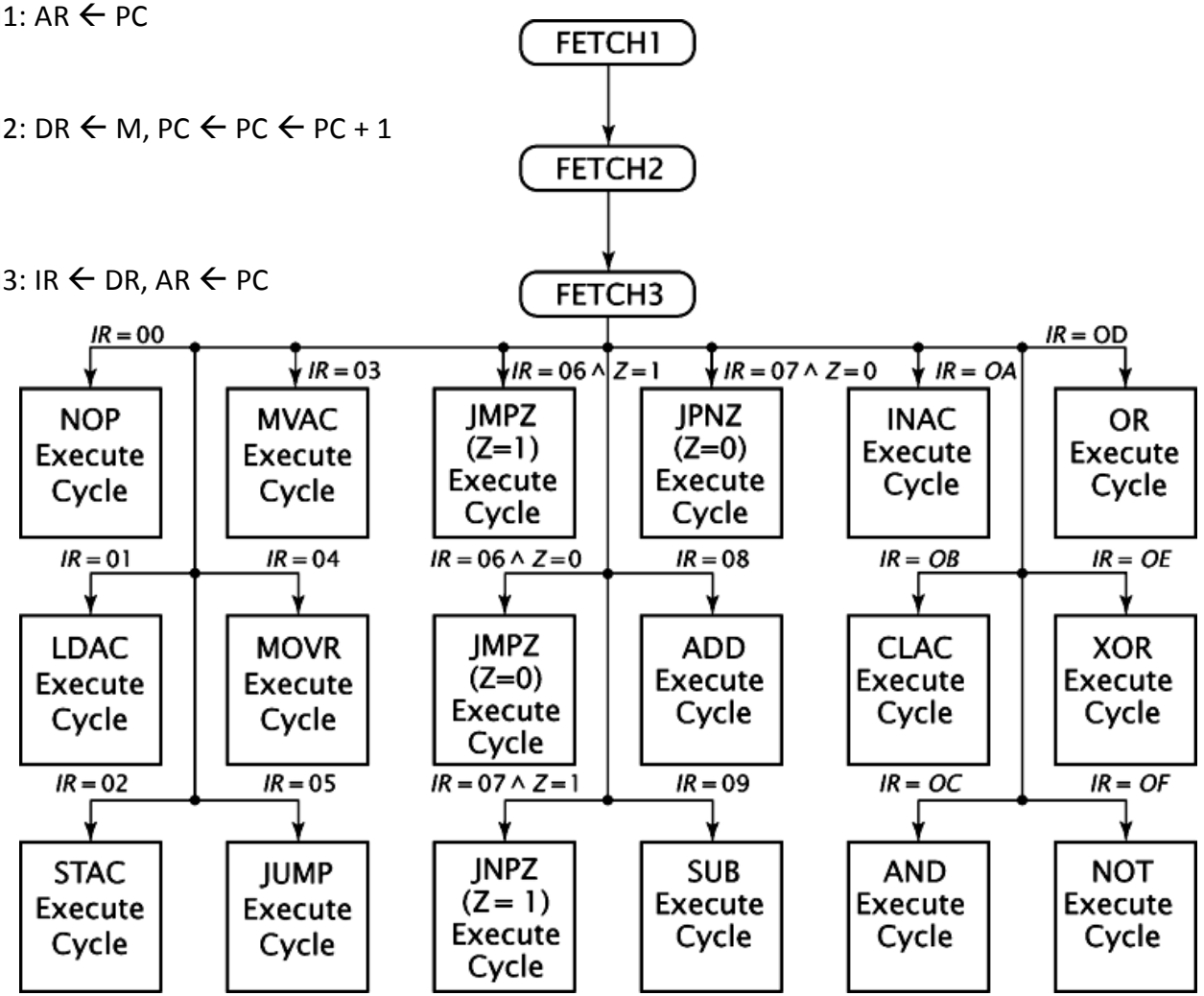
Instruction	Instruction Code	Operation
NOP	0000 0000	No Operation
LDAC	0000 0001 Γ	AC ,-- M[Γ]
STAC	0000 0010 Γ	M[Γ] ← AC
MVAC	0000 0011	R ← AC
MOVR	0000 0100	AC ← R
JUMP	0000 0101 Γ	GOTO Γ
JMPZ	0000 0110 Γ	IF(Z=1) THEN GOTO Γ
JPNZ	0000 0111 Γ	IF(Z=0) THEN GOTO Γ
ADD	0000 1000	AC ← AC + R, IF(AC + R = 0) THEN Z ← 1 ELSE Z ← 0
SUB	0000 1001	AC ← AC - R, IF(AC - R = 0) THEN Z ← 1 ELSE Z ← 0
INAC	0000 1010	AC ← AC + R, IF(AC + R = 0) THEN Z ← 1 ELSE Z ← 0
CLAC	0000 1011	AC ← 0, Z ← 1
AND	0000 1100	AC ← AC ^ R, IF(AC ^ R = 0) THEN Z ← 1 ELSE Z ← 0
OR	0000 1101	AC ← AC | R, IF(AC | R = 0) THEN Z ← 1 ELSE Z ← 0
XOR	0000 1110	AC ← AC -| R, IF(AC -| R = 0) THEN Z ← 1 ELSE Z ← 0
NOT	0000 1111	AC ← AC`, IF(AC` = 0) THEN Z ← 1 ELSE Z ← 0

Fetch and Decode Cycles for Relatively Simple CPU

FETCH1: AR ← PC

FETCH2: DR ← M, PC ← PC ← PC + 1

FETCH3: IR ← DR, AR ← PC



Instructions

LDAC1: DR ← M, PC ← PC + 1, AR ← AR + 1
LDAC2: TR ← DR, DR ← M, PC ← PC + 1
LDAC3: AR ← DR, TR
LDAC4: DR ← M
LDAC5: AC ← DR

STAC1: DR ← M, PC ← PC + 1, AR ← AR + 1
STAC2: R ← DR, DR ← M, PC ← PC + 1
STAC3: AR ← DR, TR
SRTAC4: DR ← AC
STAC5: M ← DR

MVAC1: R ← AC

MOVR1: AC ← R

JUMP1: DR ← M, AR ← AR + 1
JUMP2: TR ← DR, DR ← M
JUMP3: PC ← DR, TR

JMPZY1: DR ← M, AR ← AR + 1
JMPZY2: TR ← DR, DR ← M
JMPZY3:PC ← DR, TR

JMPZN1: PC ← PC + 1
JMPZN2: PC ← PC + 1

JPNZY1: DR ← M, AR ← AR + 1
JPNZY2: TR ← DR, DR ← M
JPNZY3:PC ← DR, TR

JPNZN1: PC ← PC + 1
JPNZN2: PC ← PC + 1

Instructions

ADD1: AC ← AC + R, IF (AC + R = 0) THEN Z ← 1 ELSE Z ← 0

SUB1: AC ← AC - R, IF (AC - R = 0) THEN Z ← 1 ELSE Z ← 0

INAC1: AC ← AC + R, IF (AC + R = 0) THEN Z ← 1 ELSE Z ← 0

CLAC1: AC ← 0, Z ← 1

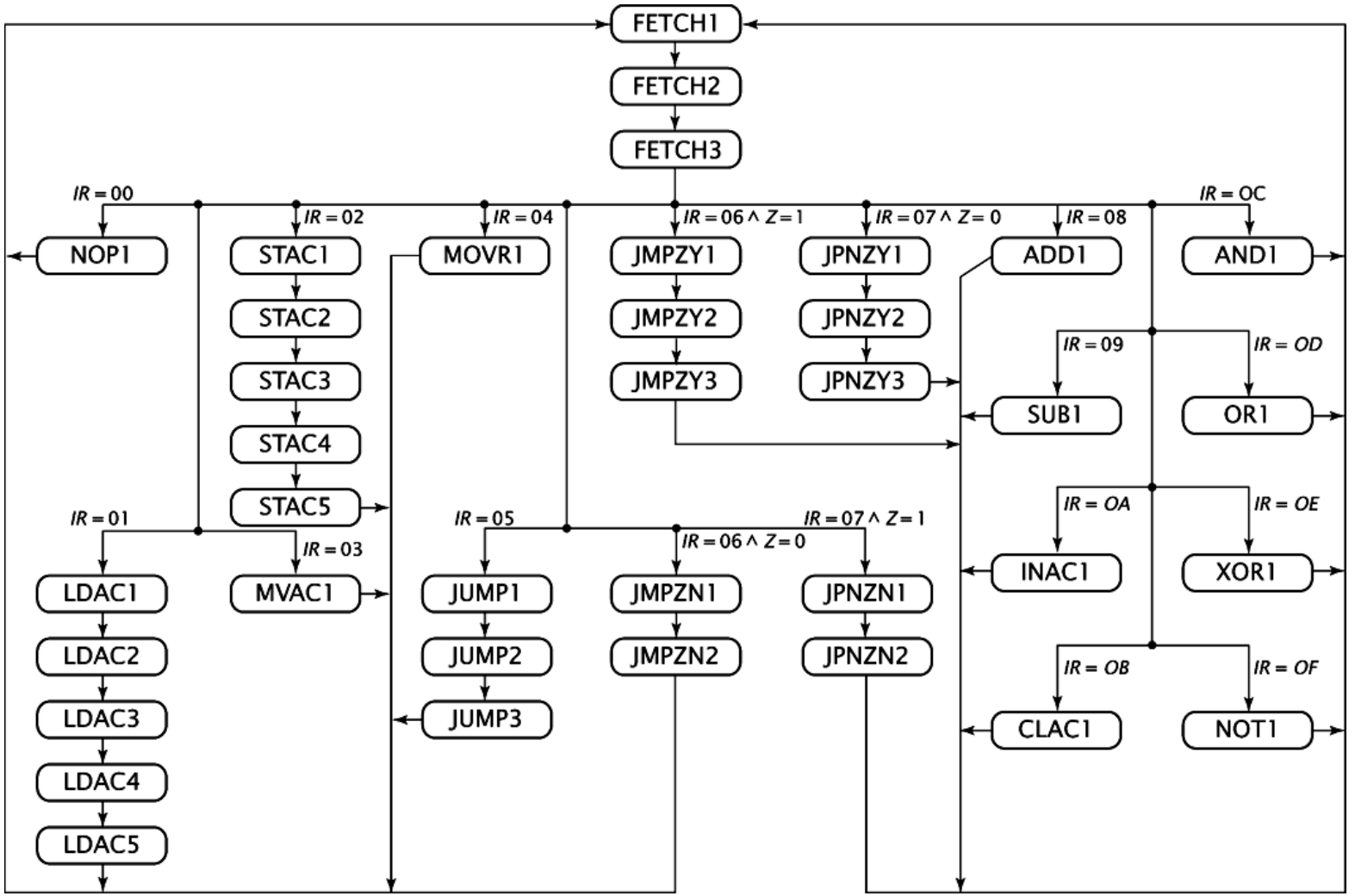
AND1: AC ← AC ^ R, IF (AC ^ R = 0) THEN Z ← 1 ELSE Z ← 0

OR1: AC ← AC | R, IF (AC | R = 0) THEN Z ← 1 ELSE Z ← 0

XOR1: AC ← AC +| R, IF (AC +| R = 0) THEN Z ← 1 ELSE Z ← 0

NOT1: AC ← AC`IF (AC` = 0) THEN Z ← 1 ELSE Z ← 0

Complete State Diagram for the Relatively Simple CPU



Establishing the Data Paths

AR: AR ← PC; AR ← AR + 1; AR ← DR, TR

PC: PC ← PC + 1; PC ← DR, TR

DR: DR ← M, DR ← AC

IR: IR ← DR

R: R ← AC

TR: TR ← DR

AC: AC ← DR; AC ← R; AC ← AC + R; AC ← R;

AC ← AC + 1; AC ← 0; AC ← AC ^ r; AC ← AC | r;

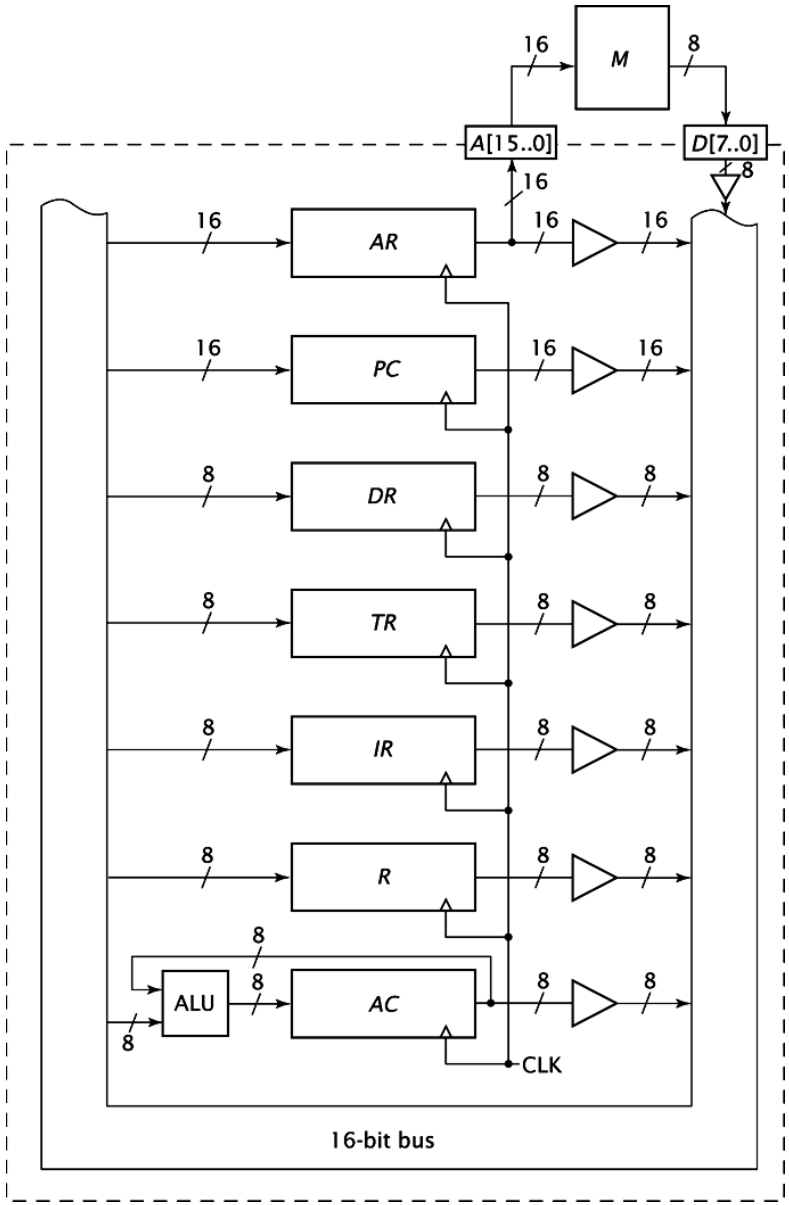
AC ← AC +| R; AC ← AC`

Z: Z ← 0 (both conditional)

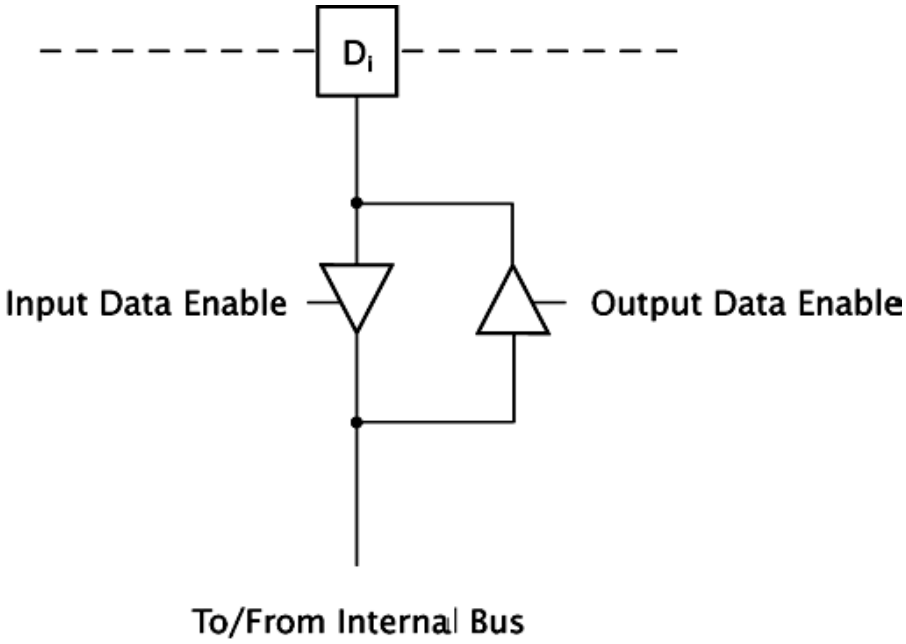
Preliminary Register Selection for the Relatively Simple CPU

Need modifications:

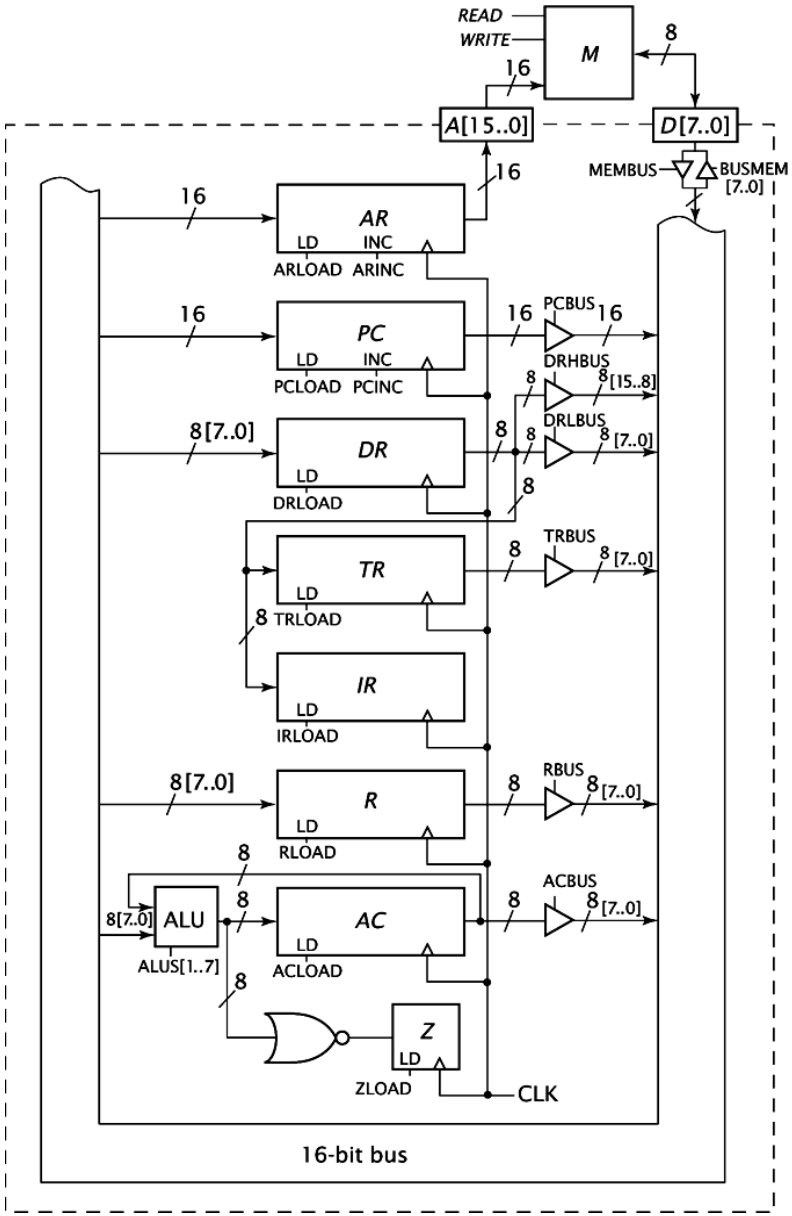
1. AR and IR do not supply data to other components
2. Pins D[7..0] are bidirectional, but cannot output data from pins in current configuration
3. 16-bit bus not fully used by al registers. Must specifiy which bits of data bus are connected to which bits of registers
4. Register Z not connected to anything



Generic Bidirectional Data Pin



Final Register Selection for the Relatively Simple CPU



Designing the Relatively Simple CPU

- Identify all transfers that modify content of AC

LDAC5: AC ← DR
MOVR1: AC ← R
ADD1: AC ← AC + R
SUB1: AC ← AC - R
INAC1: AC ← AC + R
CLAC1: AC ← 0
AND1: AC ← AC ^ R
OR1: AC ← AC | R
XOR1: AC ← AC +| R
NOT1: AC ← AC`

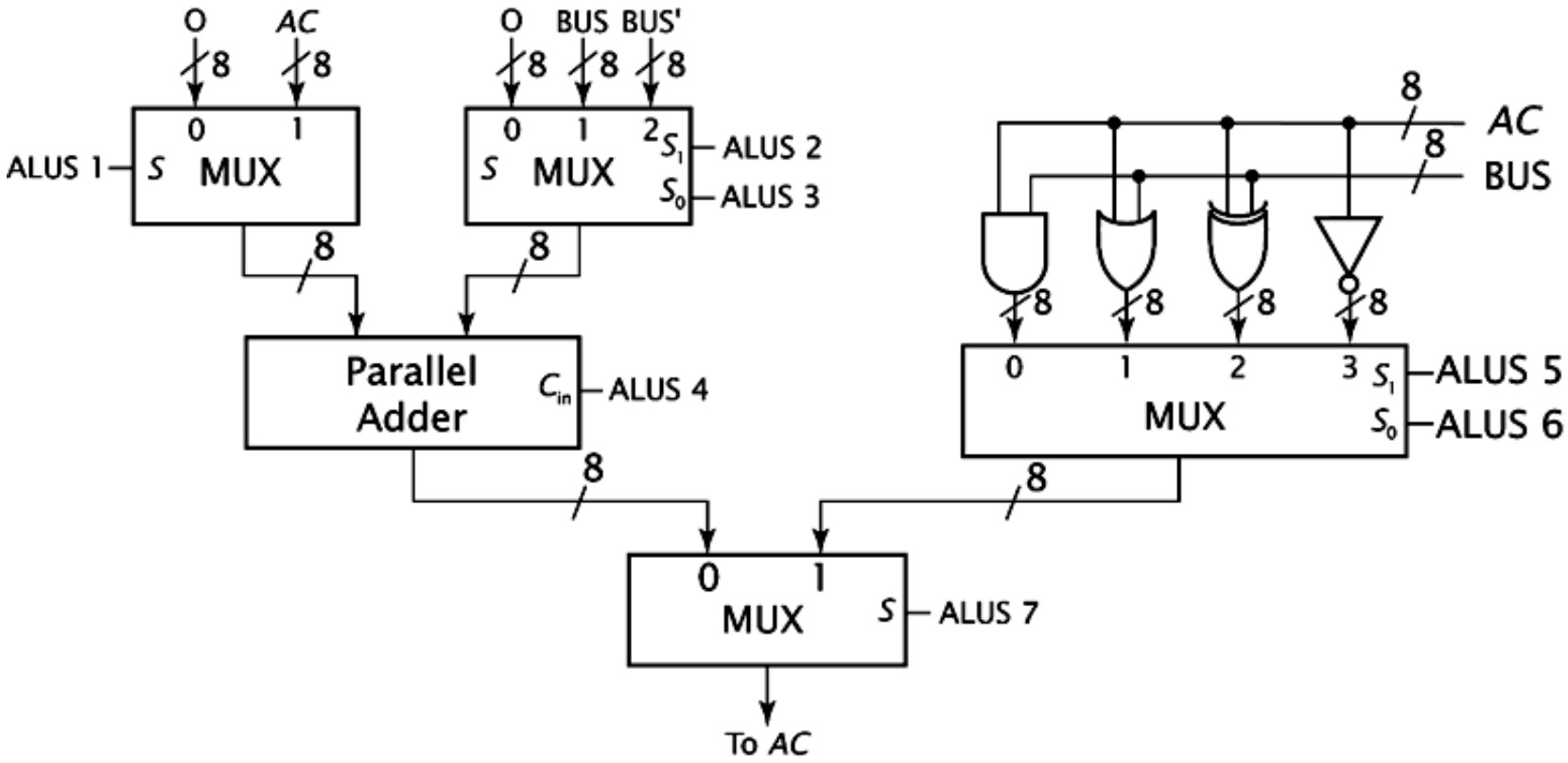
- Identify source of their operands

LDAC5: AC ← BUS
MOVR1: AC ← BUS
ADD1: AC ← AC + BUS
SUB1: AC ← AC - BUS
INAC1: AC ← AC + 1
CLAC1: AC ← 0

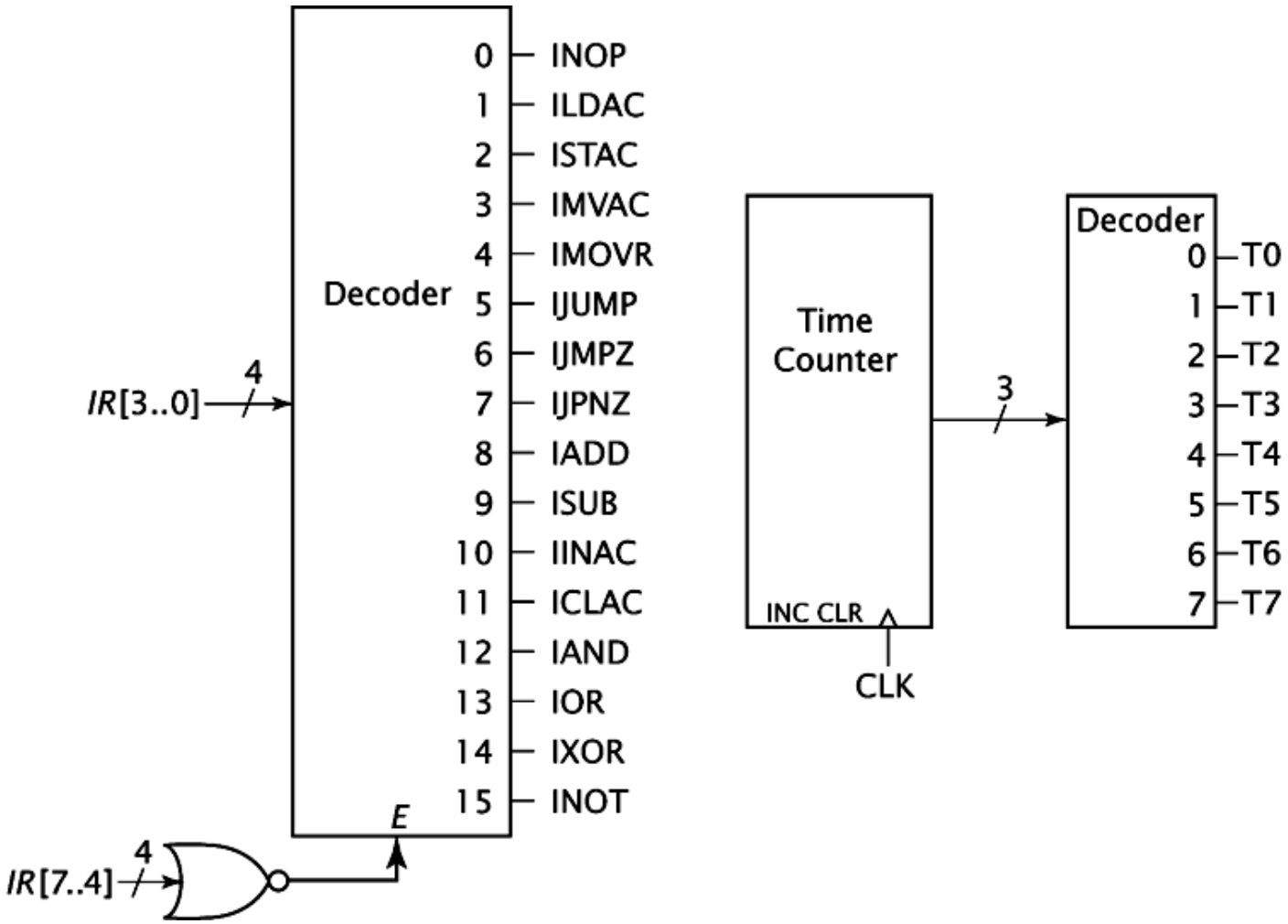
- Rewrite as sum of two values and a carry

LDAC5: AC ← 0 + BUS+ 0
MOVR1: AC ← 0 + BUS + 0
ADD1: AC ← AC + BUS + 0
SUB1: AC ← AC + BUS` + 1
INAC1: AC ← AC + 0 + 1
CLAC1: AC ← 0 + 0 + 0

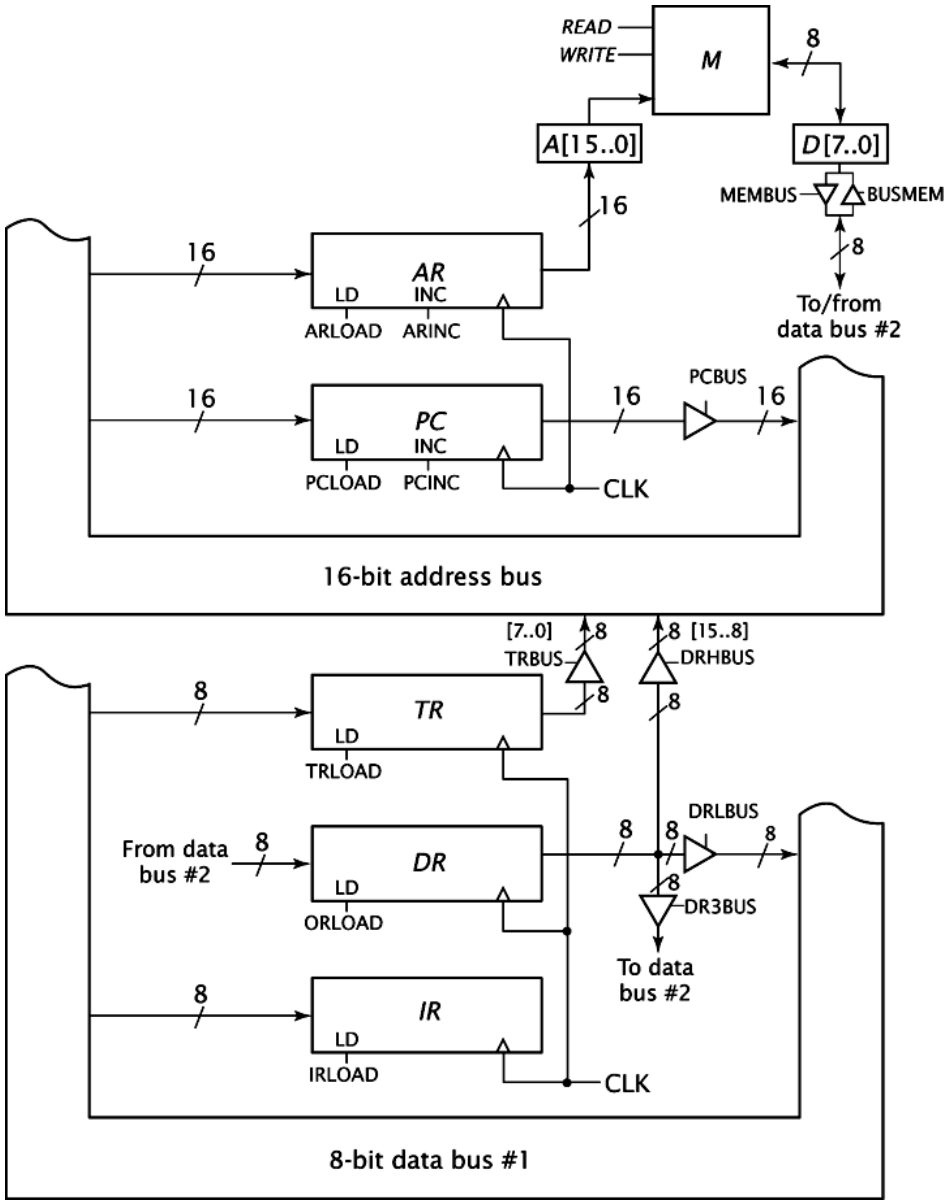
Relatively Simple ALU



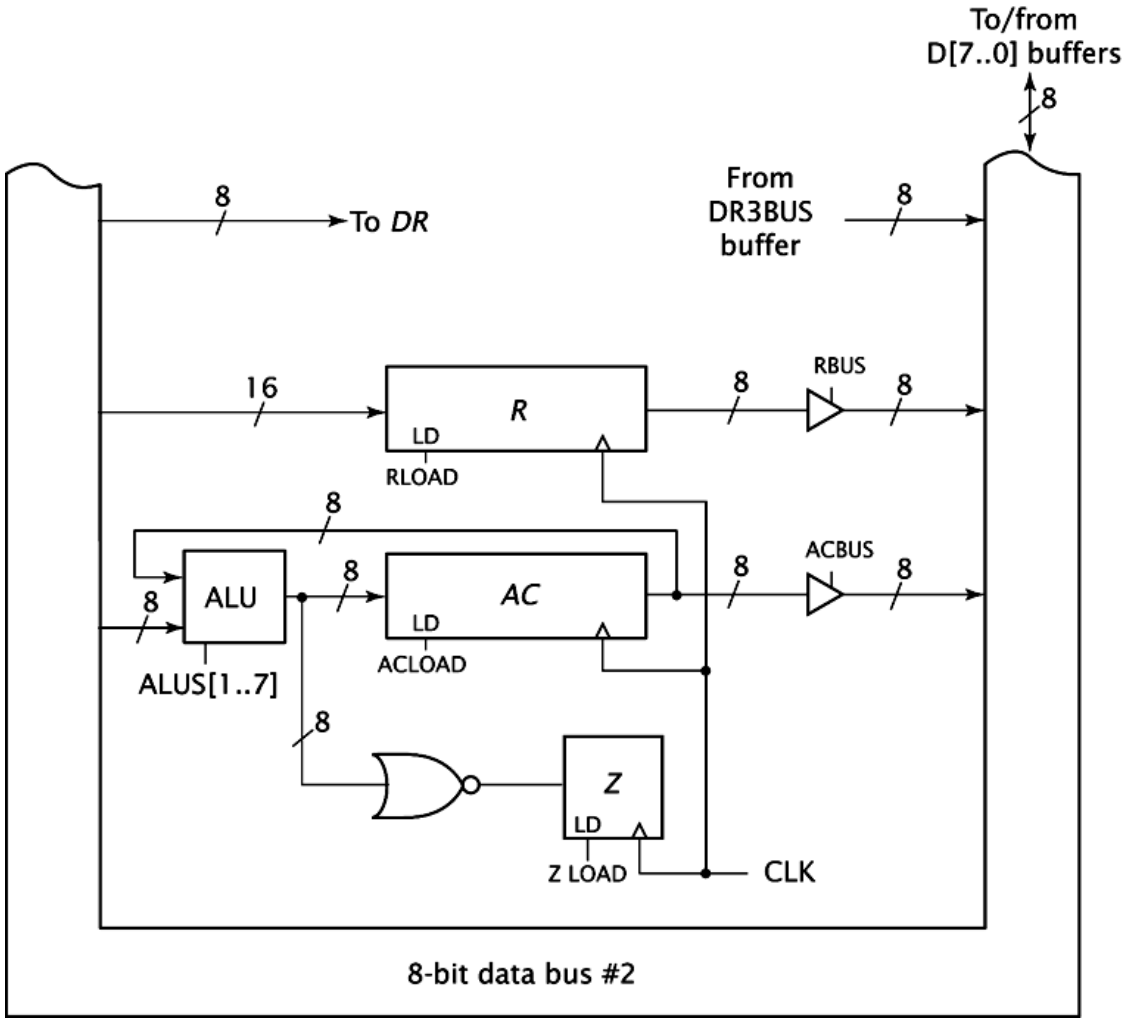
Hardwired Control Unit for the Relatively Simple CPU



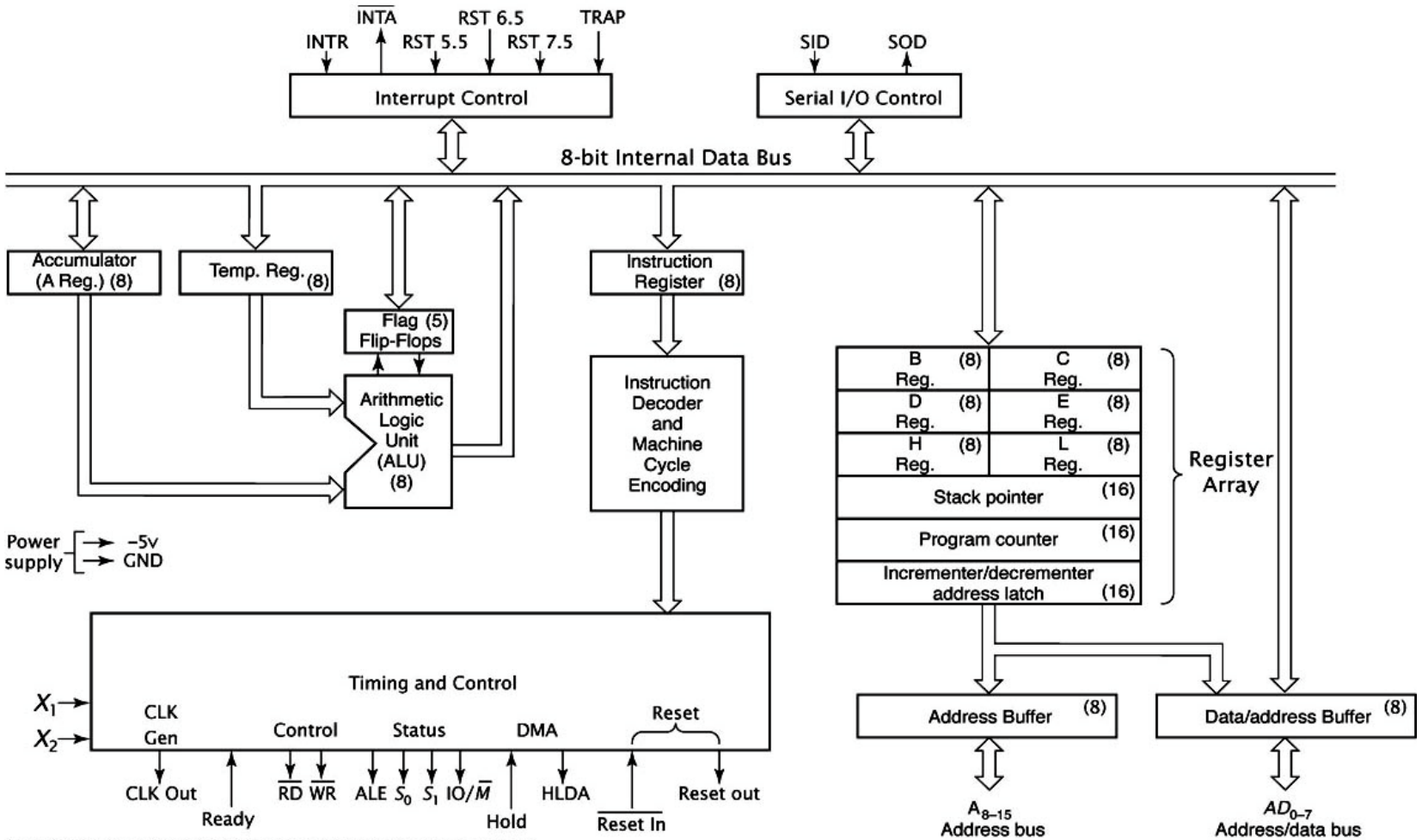
Register Selection for the Relatively Simple CPU using Multiple Buses



Register Selection for the Relatively Simple CPU using Multiple Buses (cont)

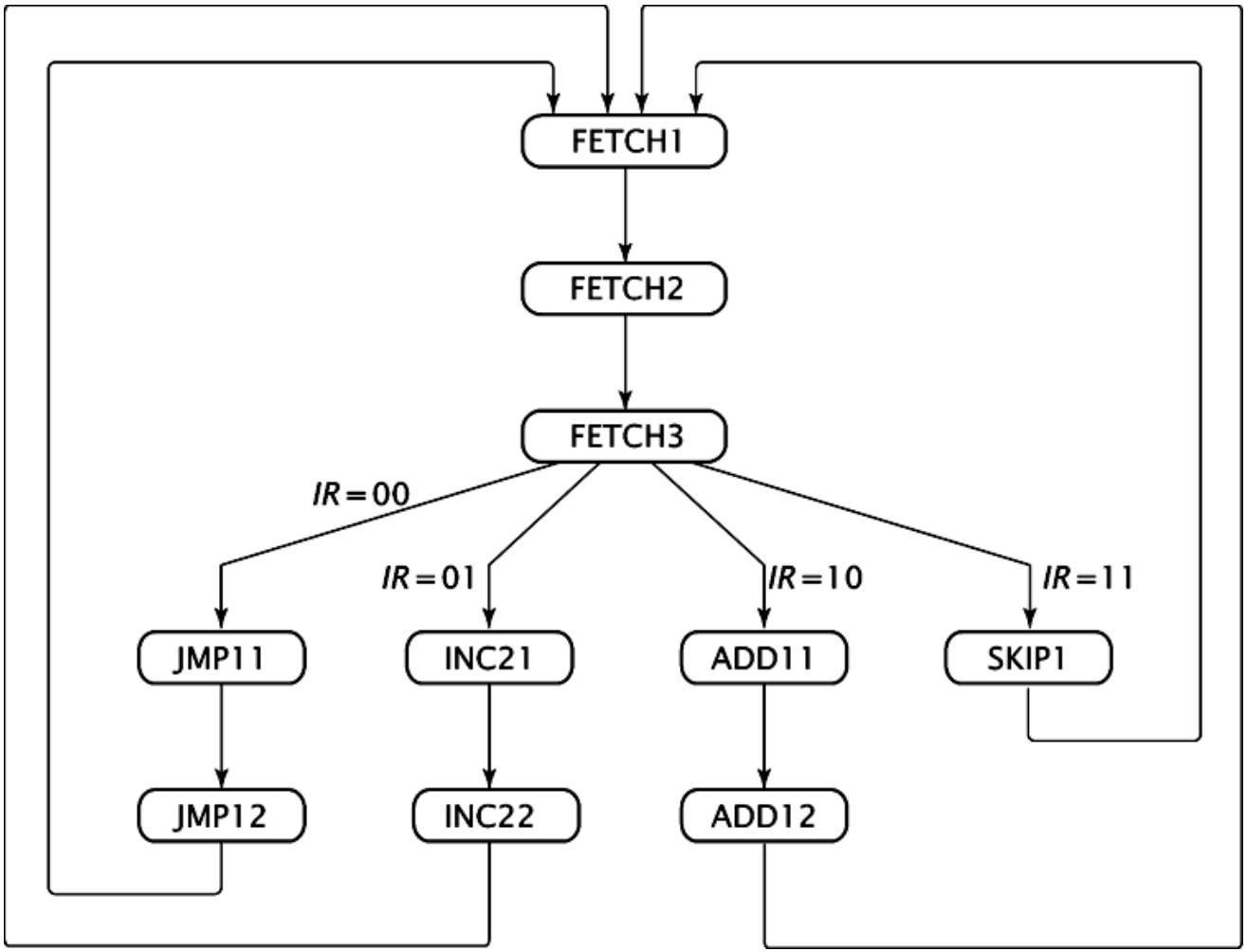


Internal Organization of the 8085 Microprocessor



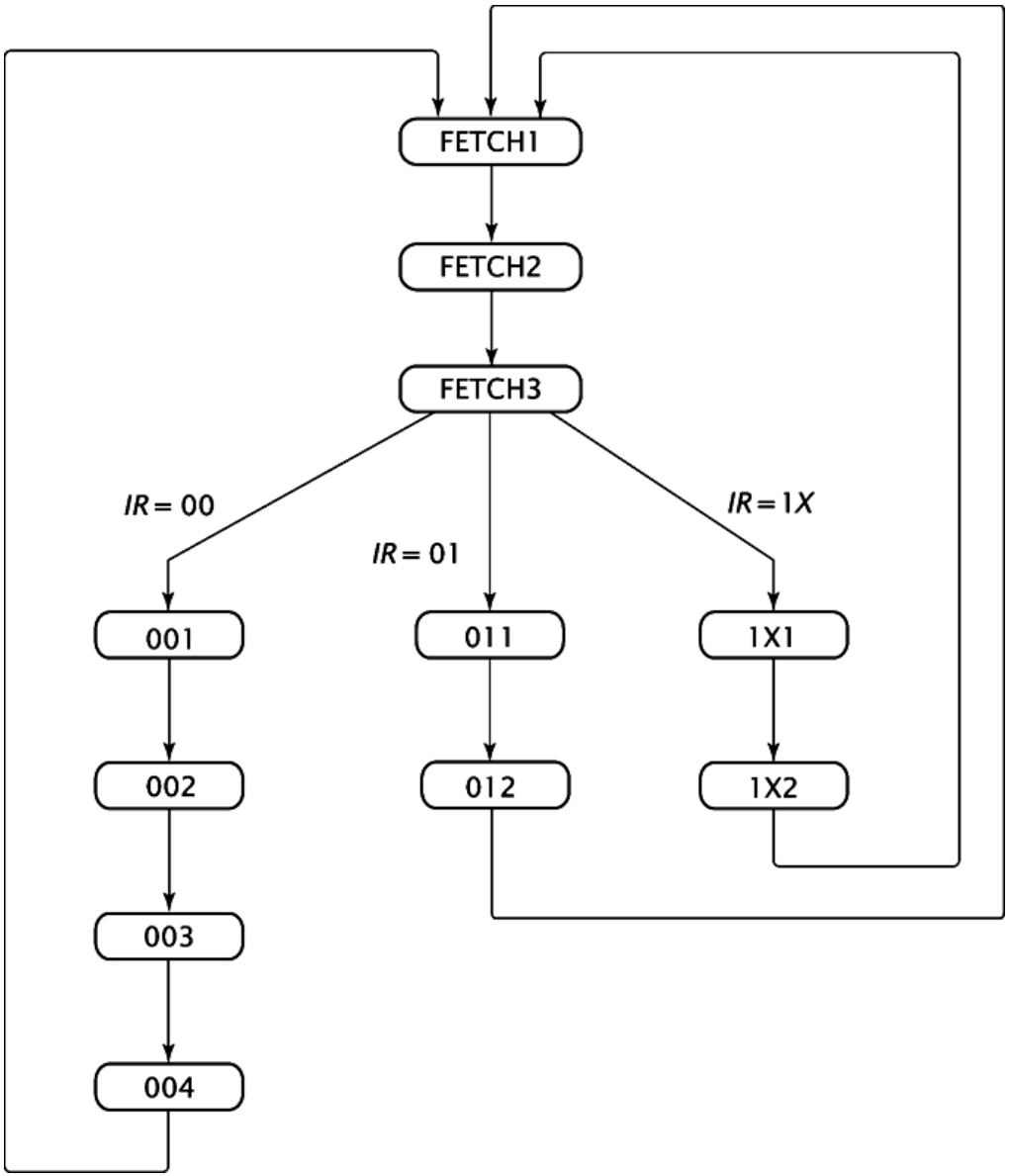
Internal organization of the 8085 microprocessor (MCS 80/85™ Family User's Manual. Reprinted by permission of Intel Corporation, Copyright Intel Corporation 1979.)

Page 260, Problem 6.1

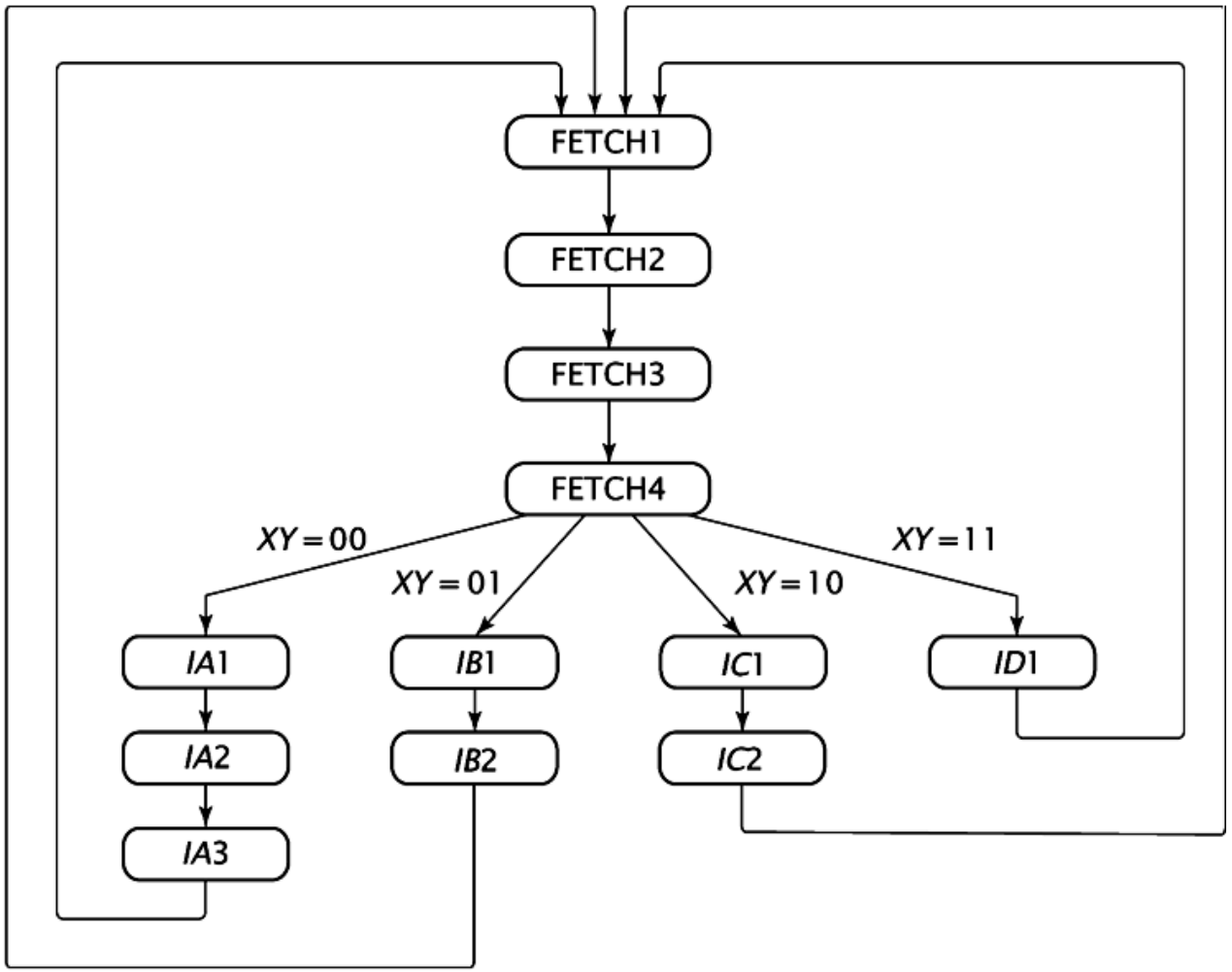


Page 261,

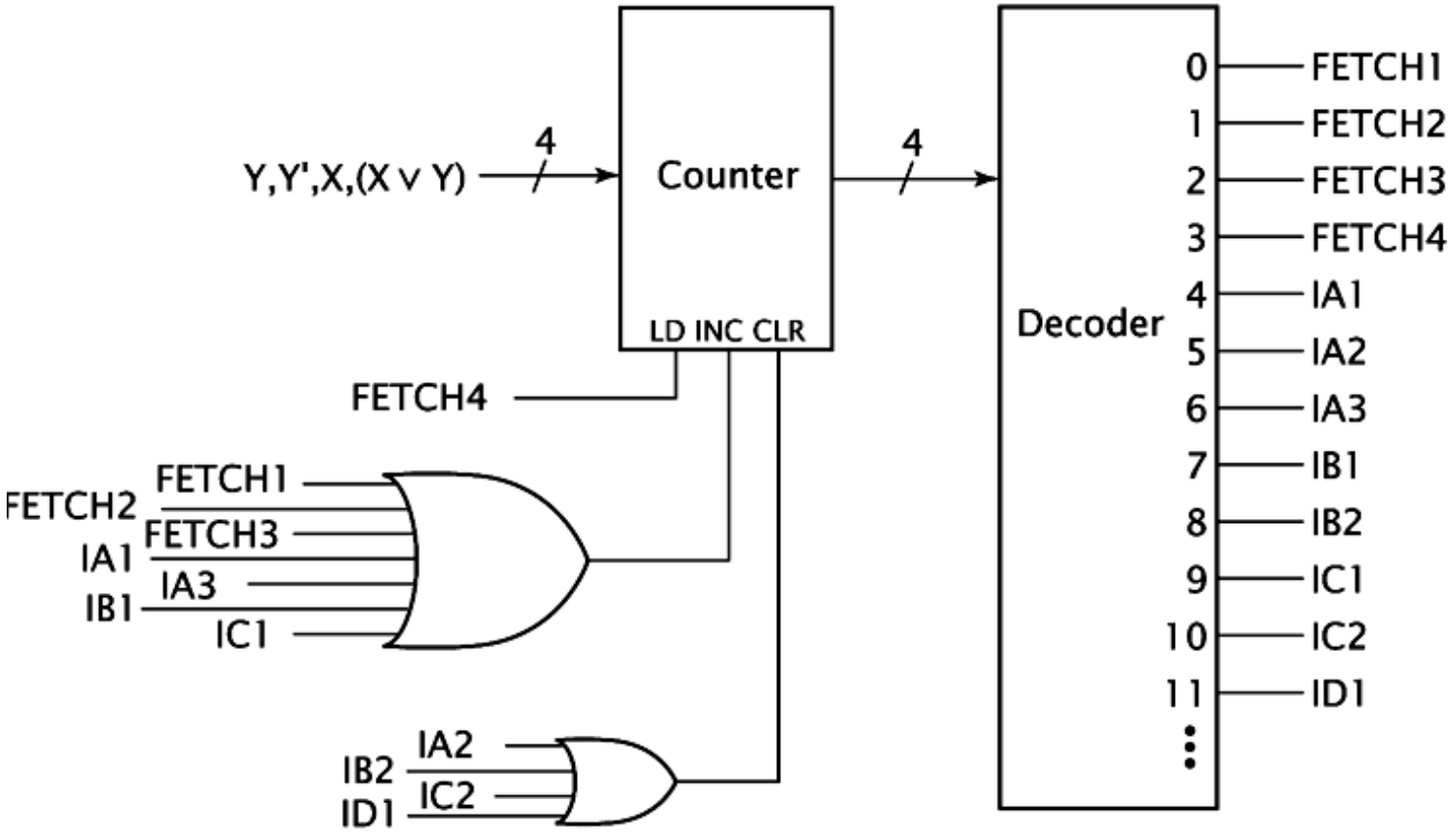
Problem 6.2



Page 262, Problem 6.4



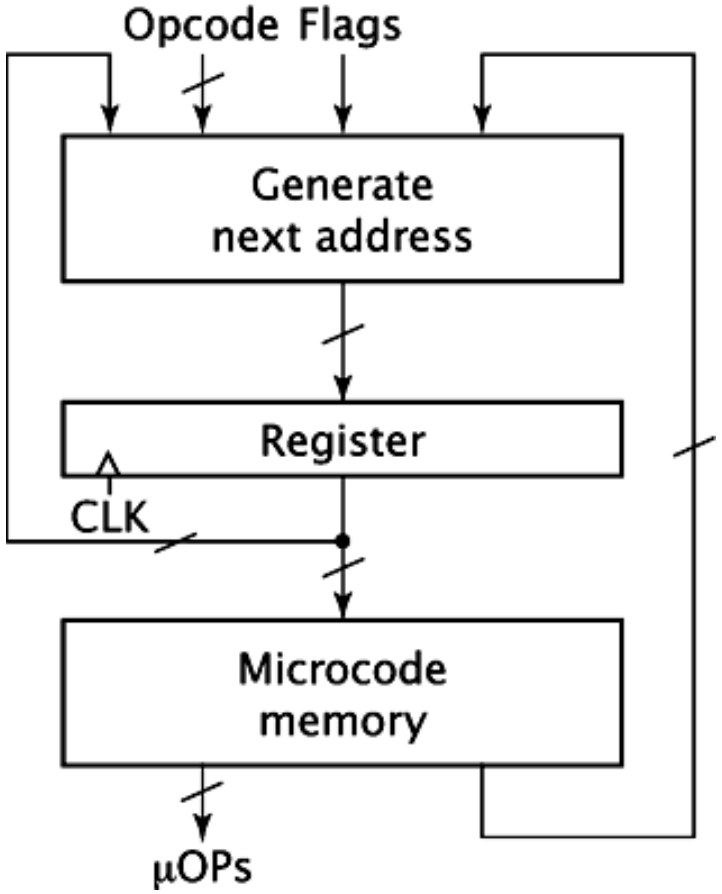
Page 262, Problem 6.4 Continued



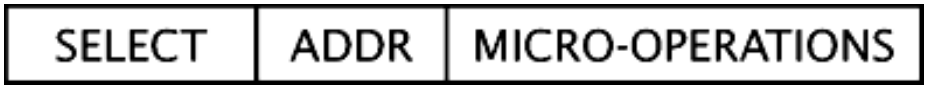
Microsequencer Control Unit Design

Generic microsequencer organiation

- µOPs: input to combinatorial logic to generate the CPU's control signals, or they directly produce the control signals (in state diagram, outputs associated with states)
- Second output is used to generate the next address
  - Input, along with opcode and flag values are used to generate address of next microinstruction (equivalent of making transition from one state to the next)
- Generate next address: some possible addresses
  - Next address in microcode memory which is the current address + 1
  - Another possible address is the absolute address supplied by microcode memory



Generic microinstruction format



- SELECT field: determines source of address of next microinstruction
- ADDR field: specifies an absolute address. Used when performing absolute jump
- MICRO-OPERATIONS field: three primary methods
  - Horizontal microcode
  - Vertical microcode
  - Direct generation of control signals

Horizontal Microcode

- List every micro operation performed by CPU
- Assign one bit in microcode for each micro operation
  - Can result in large microcode (for example, for a CPU with 50 microinstruction, that would be a microinstruction with 50 bits – pretty large
- Need a more efficient operation

Vertical Microcode

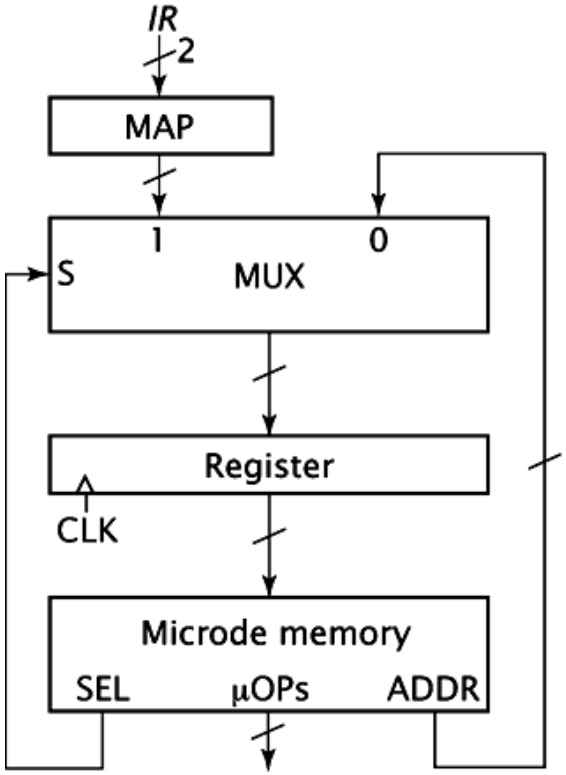
- Micro operations are grouped into fields
  - Each micro operation assigned a unique encoded value to this field
    - 16 micro operations could be encoded using four bits, with each micro operation assigned a unique binary field value from 0000 to 1111 (one would be need for a no op operation)
- Requires fewer bits for each micro operation but bust include a decoder.
- Both vertical and horizontal, CPU must convdert micro operation signal to the control signals that load, clear, and increment registers

Direct Generation of Control Signals

- Does away with micro operations
- Stores values of control signals directly in microinstruction
  - One bit connected directly to load signal, another to increment input of program counter, etc.
- Doesn’t require additional logic, but is less readable and more difficult to debug

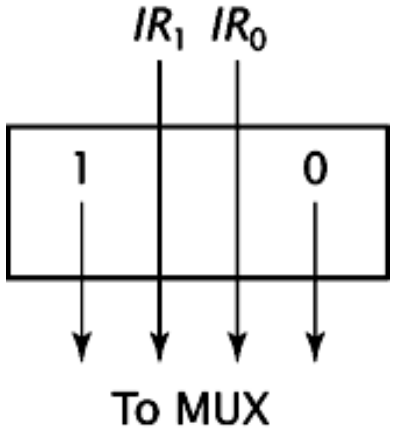
Microsequencer for Very Simple CPU

- Two possible next addresses
  - Opcode mapping
  - Absolute jump
- MUX: use to select the correct address
- Input to MUX, Register, and Microde memory and ADDR output from Microde memory is 4 bits – min bits need to determine one of 9 states of CPU



Mapping logic for the Very Simple Microsequencer

State	Address
FETCH1	0000 (0)
FETCH2	0001 (1)
FETCH3	0010 (2)
ADD1	1000 (8)
ADD2	1001 (9)
AND1	1010 (10)
AND2	1011 11)
JMP1	1100 (12)
INC1	1110 (14)



Assign each state of the finite state machine to an address in microcode

By choosing the addresses wisely we can minimize connections

Partial Microcode for the Very Simple Microsequencer

State	Address	SEL	ADDR
FETCH1	0000 (0)	0	0001
FETCH2	0001 (1)	0	0010
FETCH3	0010 (2)	1	xxxx
ADD1	1000 (8)	0	1001
ADD2	1001 (9)	0	0000
AND1	1010 (10)	0	1011
AND2	1011 11)	0	0000
JMP1	1100 (12)	0	0000
INC1	1110 (14)	0	0000

- SEL = 0 will get next address from ADDR field
- FETCH3 must map to correct execute routine, so SEL = 1 to use that address

Micro operations and their mnemonics for the Very Simple Microsequencer

- Need to list every micro operation so we can develop the code.
- The Very Simple CPU has nine micro operations.
- Need one bit for each operation.

Mnemonics	Micro Operation
ARPC	AR ← PC
ARDR	AR ← DR[5..0]
PCIN	PC ← PC + 1
PRDR	PC ← DR[5..0]
DRM	DR ← M
IRDR	IR ← DR[7..6]
PLUS	AC ← AC + DR
AND	AC ← AC ^ DR
ACIN	AC ← AC + 1

Establishing Required Data Paths

- Operations associated with each state of the CPU are:

  FETCH1: AR ← PC

  FETCH2: DR ← M, PC ← PC + 1

  FETCH3: IR ← DR[7..6], AR ← DR[5..0]

  ADD1: DR ← M

  ADD2: AC ← AC + DR

  AND1: DR ← M

  AND2: AC ← AC ^ DR

  JMP1: PC ← DR[5..0]

  INC1: AC ← AC + 1

Preliminary horizontal microcode for the Very Simple Microsequencer

State	Address	SEL	ARPC	ARDR	PCIN	PCDR	DRM	IRDR	PLUS	AND	ACIN	ADDR
FETCH1	0000 (0)	0	1	0	0	0	0	0	0	0	0	0001
FETCH2	0001 (1)	0	0	0	1	0	1	0	0	0	0	0010
FETCH3	0010 (2)	1	0	1	0	0	0	1	0	0	0	xxxx
ADD1	1000 (8)	0	0	0	0	0	1	0	0	0	0	1001
ADD2	1001 (9)	0	0	0	0	0	0	0	1	0	0	0000
AND1	1010 (10)	0	0	0	0	0	1	0	0	0	0	1011
AND2	1011 11)	0	0	0	0	0	0	0	0	1	0	0000
JMP1	1100 (12)	0	0	0	0	1	0	0	0	0	0	0000
INC1	1110 (14)	0	0	0	0	0	0	0	0	0	1	0000

Optomize

- For all states, ARDR and IRDR have the same value
- Don’t need two outputs
- Use one output to drive both micro operations, AIDR, combining
  - AR ← DR[5..0] and IR ← DR[7..6]

Optimized horizontal microcode for the Very Simple Microsequencer

State	Address	SEL	ARPC	AIDR	PCIN	PCDR	DRM	PLUS	AND	ACIN	ADDR
FETCH1	0000 (0)	0	1	0	0	0	0	0	0	0	0001
FETCH2	0001 (1)	0	0	0	1	0	1	0	0	0	0010
FETCH3	0010 (2)	1	0	1	0	0	0	0	0	0	xxxx
ADD1	1000 (8)	0	0	0	0	0	1	0	0	0	1001
ADD2	1001 (9)	0	0	0	0	0	0	1	0	0	0000
AND1	1010 (10)	0	0	0	0	0	1	0	0	0	1011
AND2	1011 11)	0	0	0	0	0	0	0	1	0	0000
JMP1	1100 (12)	0	0	0	0	1	0	0	0	0	0000
INC1	1110 (14)	0	0	0	0	0	0	0	0	1	0000

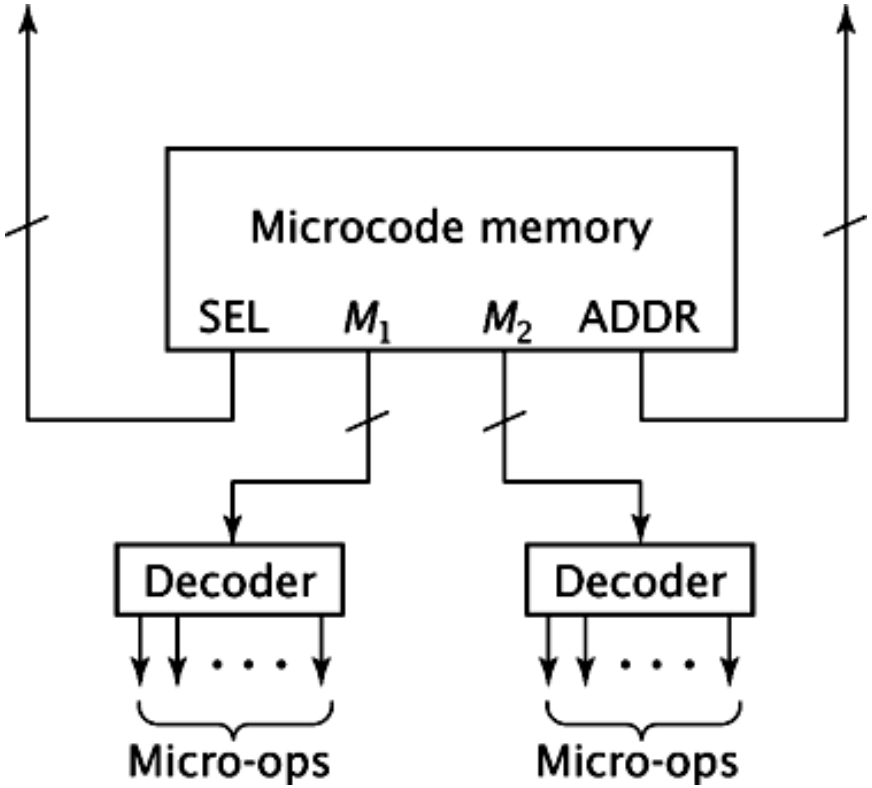
Control signal values for the Very Simple CPU

From the previous table the control signals can be generated

Signal	Value
ARLOAD	ARPC v AIDR
PCLOAD	PCDR
PCINC	PCIN
DRLOAD	DRM
ARLOAD	PLUS v AND
ARIINC	ACIN
IRLOAD	AIDR
ALUSELL	AND
MEMBUS	DRM
PCBUS	ARPC
DRBUS	AIDR v PCDR v PLUS v AND
READ	DRM

Generic generation fo micro operations from vertical microcode

- Vertical code table consists mostly of zeros – 85%
- Vertical micro operations are grouped into fields
- Can use decoders to generate the instructions



Guidelines to design microsequencer using vertical microcode

1. Whenever two micro operations occur during the same state, assign them to different fields.
2. Include a NOP in each field if necessary.
   - Needed because some value must be output every cycle
3. Distribute the remaining micro operations to make best use of the micro operation field bits.
4. Group together micro operations that modify the same registers in the same fields.

Find micro operations that can run concurrently

- DRM and PCIN both occur in FETCH2, must be assigned to different fields
  - Will need at least two fields for the Very Simple CPU

M1	M2
NOP	NOP
DRM	PCIN

- Since PCIN and PCDR both modify PC, add PCDR to M2
- Then arbitrarily assign the remaing micro operations to the fields, keeping micro operations that change the same register in the same field.

Micro operation field assignments and values

MI	
Value	Micro Operation
000	NOP
001	DRM
010	ARPC
011	AIDR
100	PCDR
101	PLUS
110	AND
111	ACIN

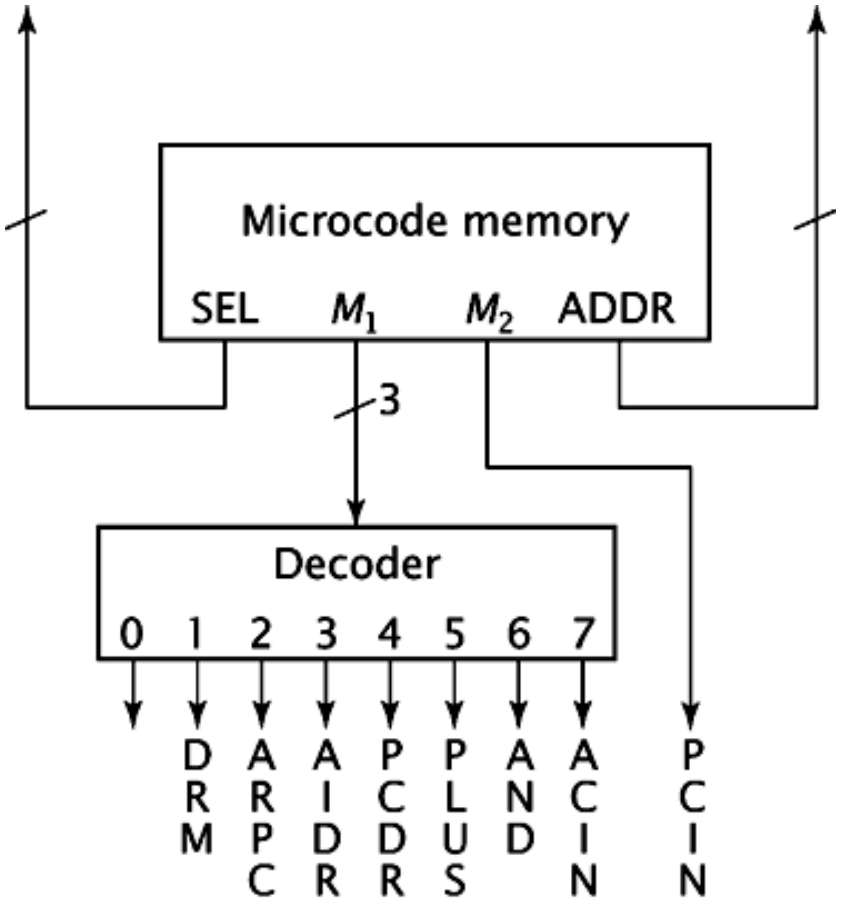
M2	
Value	Micro Operation
0	NOP
1	PCIN

- To optimize, it is best to maximize to a power of two, so is possible, put eight micro operations in M1 and two in M2.
- M1 requires 3 bits, M2 only one bit, for total of 4 bits

Vertical microcode for the Very Simple Microsequencer

State	Address	SEL	M1	M2	ADDR
FETCH1	0000 (0)	0	010	0	0001
FETCH2	0001 (1)	0	001	0	0010
FETCH3	0010 (2)	1	011	1	xxxx
ADD1	1000 (8)	0	001	0	1001
ADD2	1001 (9)	0	101	0	0000
AND1	1010 (10)	0	001	0	1011
AND2	1011 11)	0	110	0	0000
JMP1	1100 (12)	0	100	0	0000
INC1	1110 (14)	0	111	0	0000

Generating micro operations for vertical microcode for the Very Simple CPU



Directly Generating Control Signals from the Microcode

- Use one bit for each control signal
  - Set to 1, it is active
  - Set to 0, it is not active
- Example

  FETCH2: DR ← M and PC ← PC + 1

  READ output data from memory

  MEMBUS to allow data onto internal bus

  DRLOAD to load data from bus into DR

  PCINC to perform second micro operation

Set those four to 1, the others to 0

Microcode to directly generate control signals for the Very Simple Microsequencer

State	Address	SEL	ARLOAD	PCLOAD	PCINC	DRLOAD	ACLOAD	ACINC	IRLOAD	ALUSEL	MEMBUS	PCBUS	DRBUS	READ	ADDR
FETCH1	0000 (0)	0	1	0	0	0	0	0	0	0	0	1	0	0	0001
FETCH2	0001 (1)	0	0	0	1	1	0	0	0	0	1	0	0	1	0010
FETCH3	0010 (2)	1	1	0	0	0	0	0	1	0	0	0	1	0	xxxx
ADD1	1000 (8)	0	0	0	0	1	0	0	0	0	1	0	0	1	1001
ADD2	1001 (9)	0	0	0	0	0	1	0	0	0	0	0	1	0	0000
AND1	1010 (10)	0	0	0	0	1	0	0	0	0	1	0	0	1	1011
AND2	1011 11)	0	0	0	0	0	1	0	0	1	0	0	1	0	0000
JMP1	1100 (12)	0	0	1	0	0	0	0	0	0	0	0	1	0	0000
INC1	1110 (14)	0	0	0	0	0	0	1	0	0	0	0	0	0	0000

Optimized microcode to directly generate control signals

State	Address	SEL	ARLOAD	PCLOAD	PCINC	DRM	ACLOAD	ACINC	IRLOAD	ALUSEL	PCBUS	DRBUS	ADDR
FETCH1	0000 (0)	0	1	0	0	0	0	0	0	0	1	0	0001
FETCH2	0001 (1)	0	0	0	1	1	0	0	0	0	0	0	0010
FETCH3	0010 (2)	1	1	0	0	0	0	0	1	0	0	1	xxxx
ADD1	1000 (8)	0	0	0	0	1	0	0	0	0	0	0	1001
ADD2	1001 (9)	0	0	0	0	0	1	0	0	0	0	1	0000
AND1	1010 (10)	0	0	0	0	1	0	0	0	0	0	0	1011
AND2	1011 11)	0	0	0	0	0	1	0	0	1	0	1	0000
JMP1	1100 (12)	0	0	1	0	0	0	0	0	0	0	1	0000
INC1	1110 (14)	0	0	0	0	0	0	1	0	0	0	0	0000

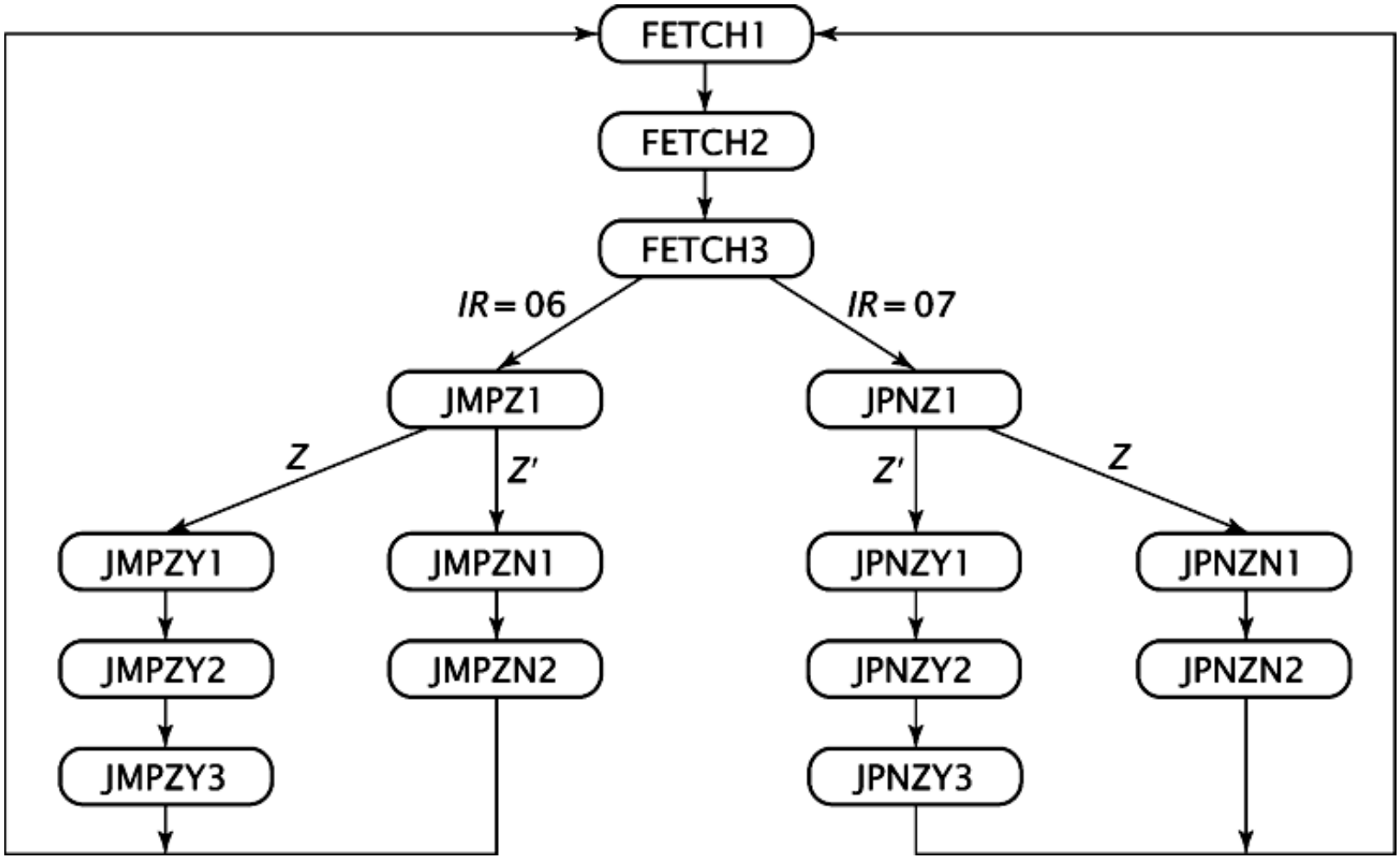
Advantage to Directly Generating Control Signals

- Does not require additional logic to convert the outputs of the microcode memory to control signals
- However, less readable and more difficult to debug

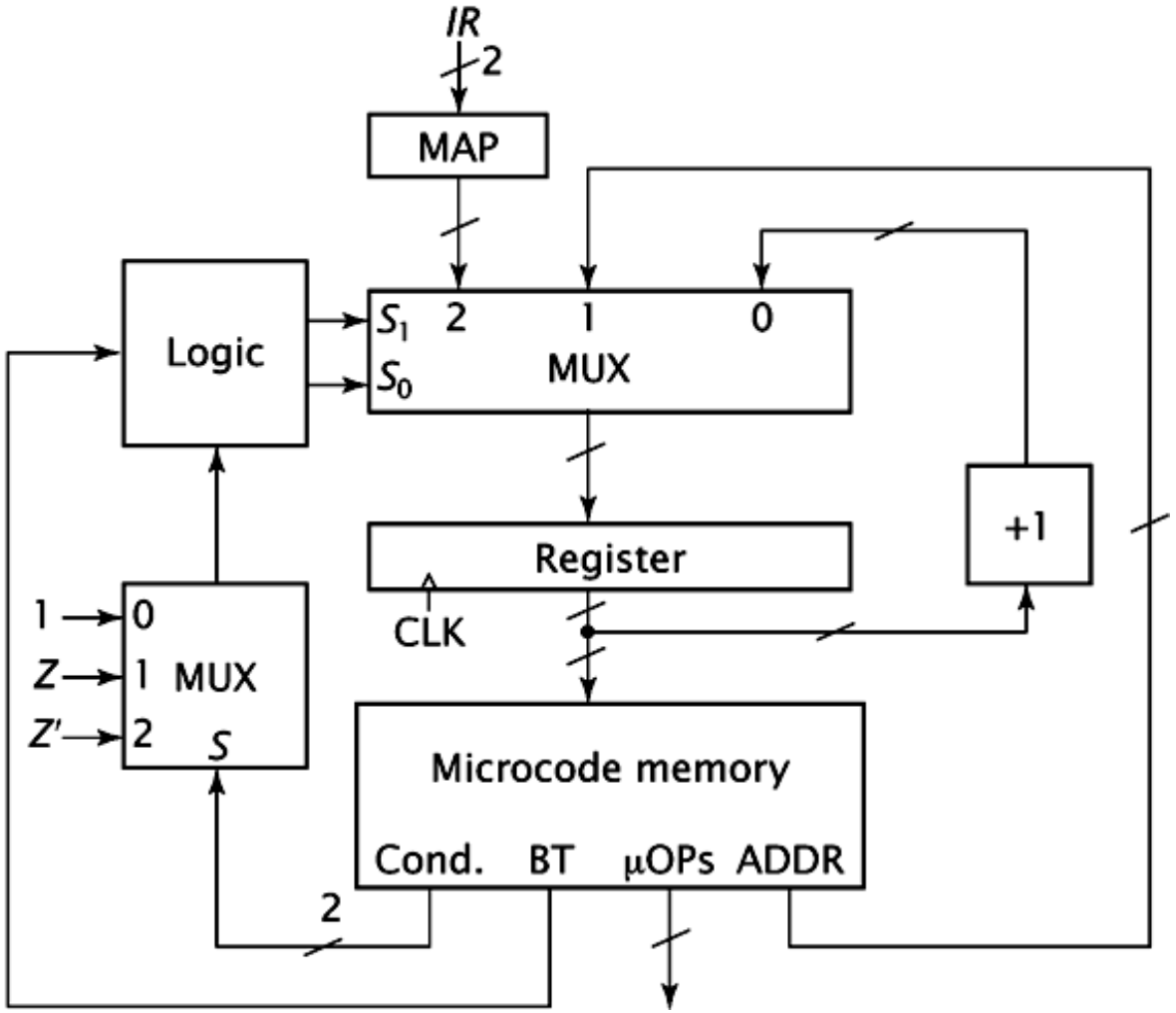
Relatively Simple CPU

- Need to modify state diagram to accommodate z flag
  - Don’t want to input z flag into mapping logic
- Instead of FETCH3 mapping to MPZY1, JMPZN1, JPNZY1, and JPNZY2, create new state, JMPZ1 and JPNZ1, and have FETCH3 map to these

Modified conditional jump execute routines



Microsequencer for the Relatively Simple CPU



State Assignments for Relatively Simple Microsequencer

State	Location
FETCH1	1
FETCH2	2
FETCH3	3
NOP1	0
LDAC1	4
LDAC2	5
LDAC3	6
LDAC4	7
LDAC5	33
STAC1	8
STAC2	9
STAT3	10
STAT4	11

State	Location
STAC5	14
MVAC1	12
MOVR1	16
JUMP1	20
JUMP2	21
JUMP3	22
JMPZ1	24
JMPZY1	25
JMPZY2	26
JMPZY3	27
JMPZN1	41
JMPZN2	42
JPNZ1	28

State	Location
JPNZY1	29
JPNZY2	30
JPNZY3	31
JPNZN1	45
JPNZN2	46
ADD1	32
SUB1	36
INAC1	40
CLAC1	44
AND1	48
OR1	52
NOR1	56
NOT1	60

Conditional Values

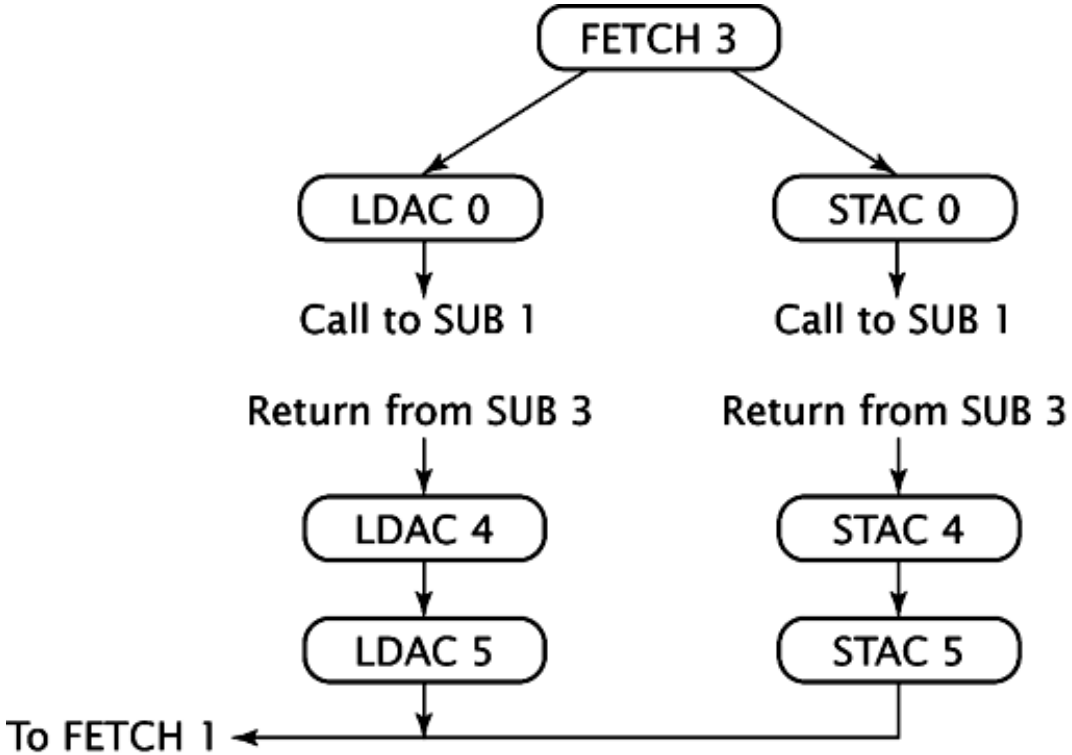
Condition	Type	S_1S_0
1	Unconditional	00
Z	Z = 1	01
Z’	Z = 0	10

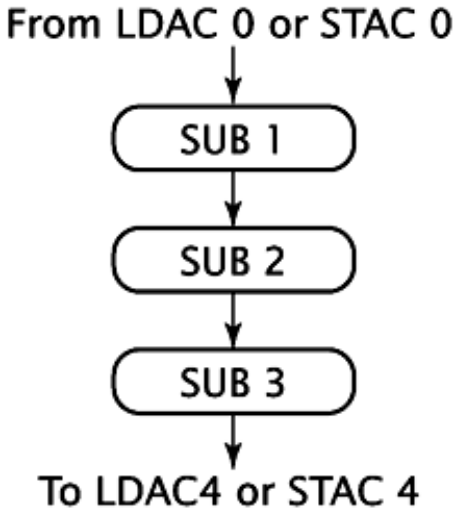
Branch Types and Branch Logic

Branch Type	BT	Branch Function
JUMP(J)	0	IF (condition) THEN Next Address = ADDR ELSE Next Address = Current Address + 1
MAP(M)	1	Next Adderess = Map

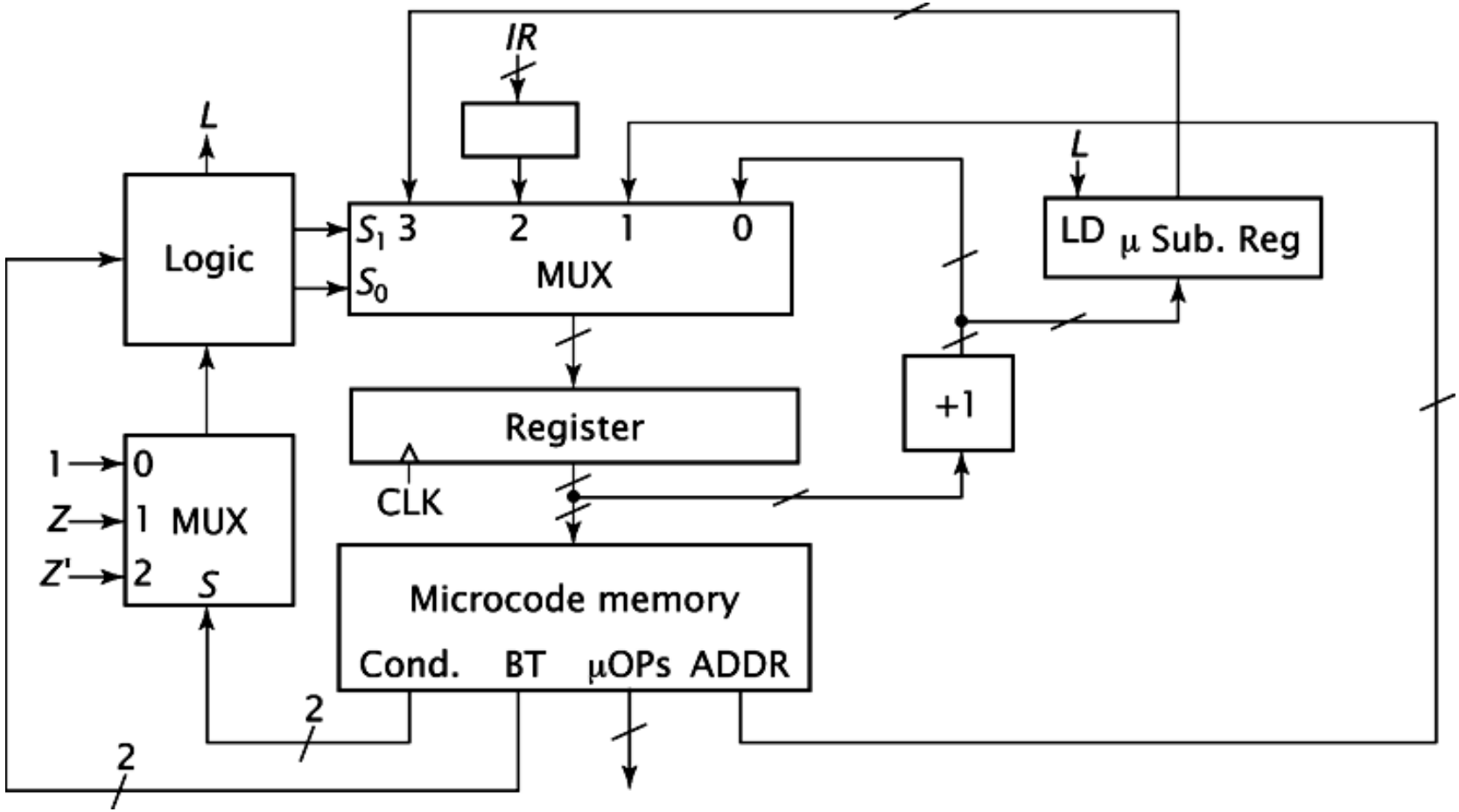
BT	Condition	Next Address	S_1S_0
0	0	Current address + 1	00
0	1	ADDR	01
1	X	MAP	10

Modified LDAC and STAC execute routines using microsubroutines

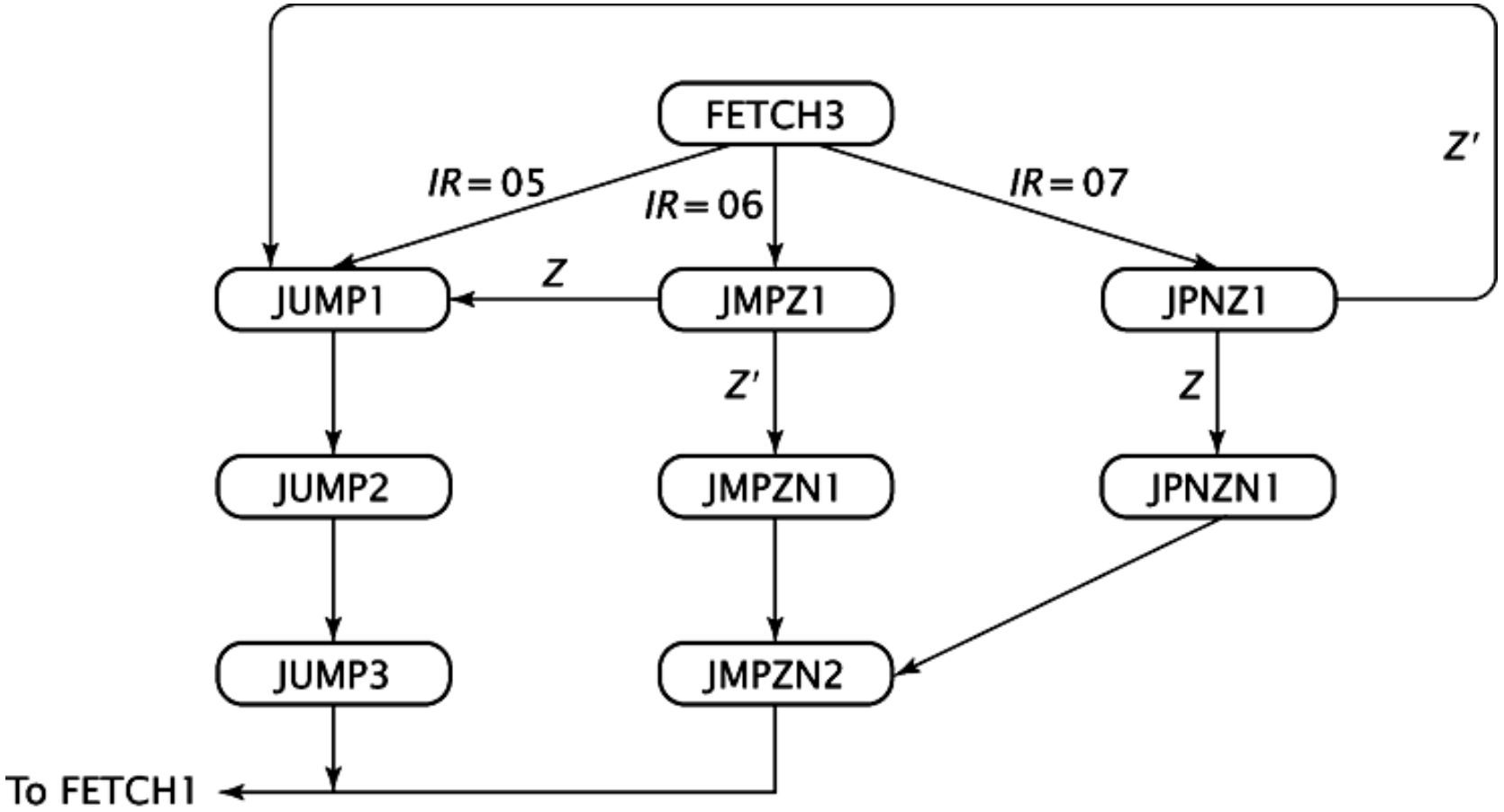




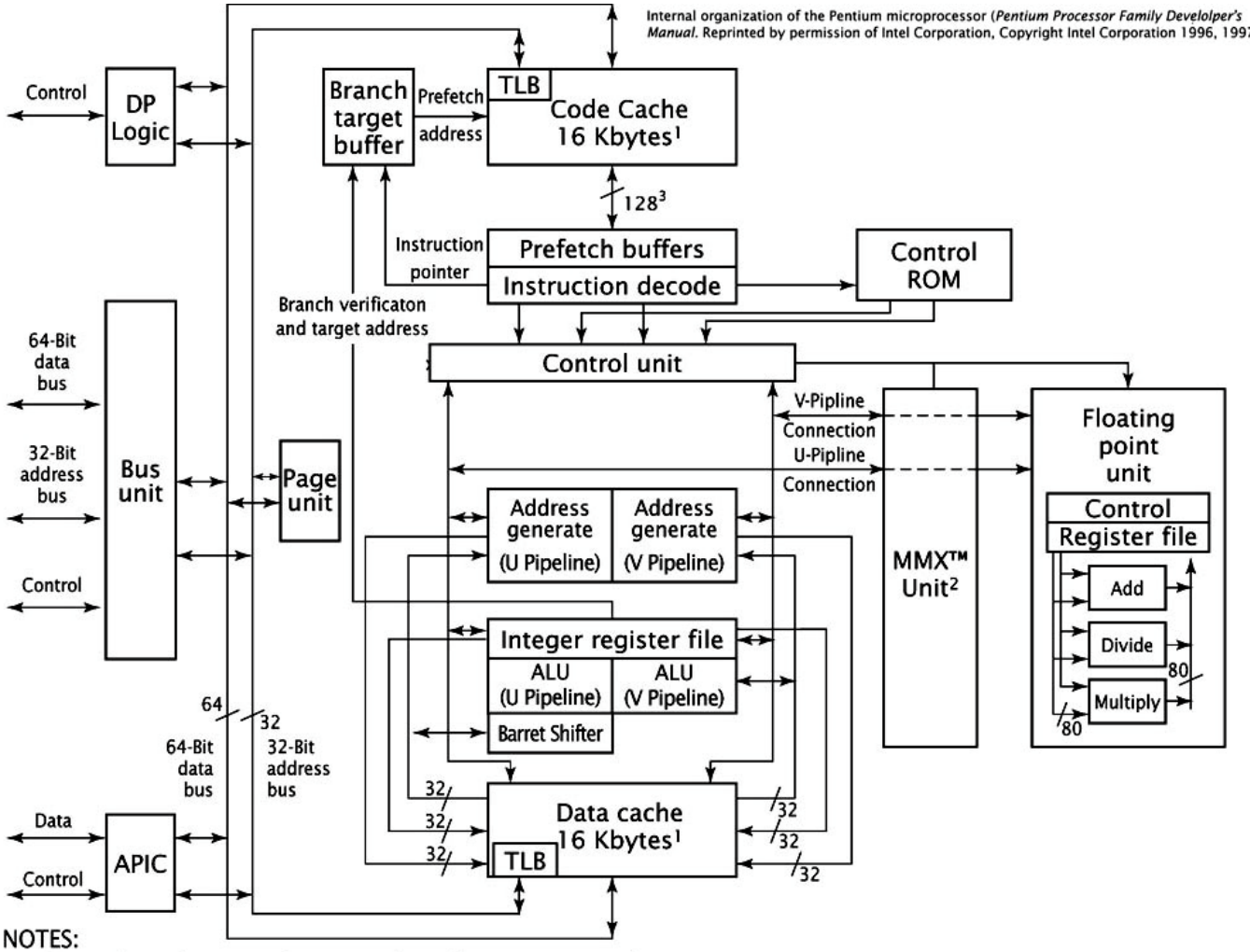
Microsequencer for the Relatively simple CPU with microsubroutines



Modified jump and conditional jump execute routines



Pentium Microprocessor



NOTES:

1. The Code and Data caches are each 8 Kbytes in size on the Pentium® processor (75/90/100/120/133/150/166/200).
2. The MMX Unit is present only on the Pentium processor with MMX™ technology.
3. The internal instruction bus is 256 bits wide on the Pentium processor (75/90/100/120/133/150/166/200).