Chapter 8: Atmel's AVR 8-bit Microcontroller
Part 3 – Microarchitecture

Prof. Ben Lee
Oregon State University
School of Electrical Engineering and Computer Science

Chapter Goals

- Understand what basic components needed and how they are interconnected to implement a datapath to perform data transfer and processing.
- Design a control unit to provide appropriate signals to the various components to control the operations within the datapath.

Contents

8.1 Microarchitecture

AVR Datapath

- Fetch - get the next instruction from memory.
- Execute - figure out what to do with it and do it.

This is how all CPUs work!

AVR Control Unit

- Makes the Fetch/Execute happen
  - Controls the fetch (of instruction)
  - Decodes instructions
  - Controls execution of instructions
- Two basic types
  - Hardwired (fast)
  - Microprogrammed (flexible)

8.2 Instruction Format

AVR Instruction Format

Two-operand format

----	--rd	dddd	rrrr

(e.g., ADD, CP, MOV, etc.)

I/O format

----	-AAd	dddd	AAAA

(e.g., IN, OUT)

Displacement format

--q-	qq-d	dddd	-qqq

(e.g., LDD, STD)

Immediate format

----	KKKK	dddd	KKKK

(e.g., LDI, ANDI, ORI, etc.)

One-operand format

----	---d	dddd	----

(e.g., INC, DEC, COMP, etc.)

PC-relative format

----	kkkk	kkkk	kkkk	k12

----	--kk	kkkk	k---	k7

(e.g., RJMP, RCALL, BRcc, etc.)

Direct format

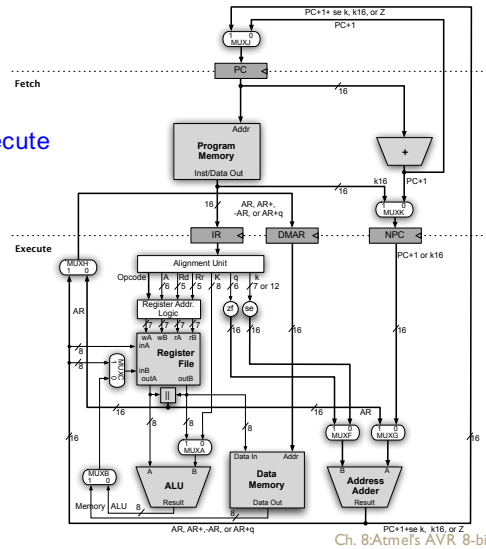
----	---k	kkkk	---k
kkkk	kkkk	kkkk	kkkk

(e.g., JMP, CALL)

8.3 Components in the Basic Datapath

Basic AVR Datapath

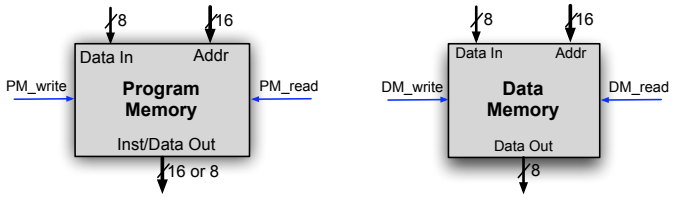
Two stages:
Fetch and Execute



Registers

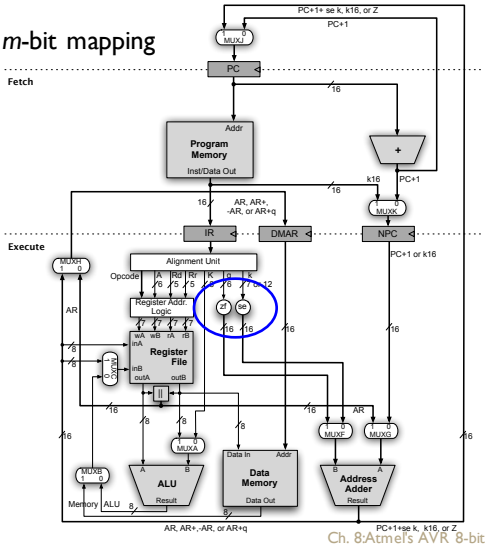
- PC – Program Counter
- IR – Instruction Register
- DMAR – Data Memory Address Register
- NPC – Next PC

Memories

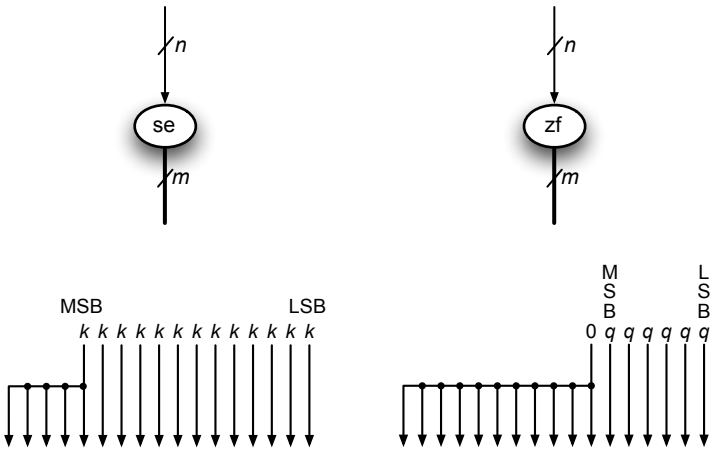


Sign-Extension & Zero-Fill

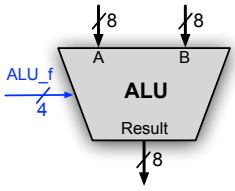
n-bit to *m*-bit mapping



Sign-Extension & Zero-Fill



ALU

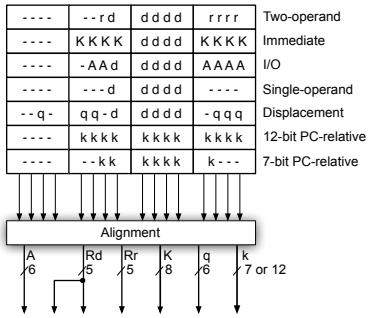


Performs 8-bit operations.

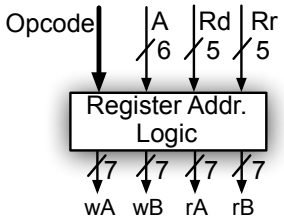
Operation	Description	ALU_f
Arithmetic Operations		
$Result = A + B$	Add	0000
$Result = A + B + C$	Add with carry (C)	0001
$Result = A + \overline{B} + 1$	Subtract	0010
$Result = \overline{B} + 1$	Negate	0011
$Result = A + 1$	Increment	0100
$Result = A - 1$	Decrement	0101
$Result = A$	Move A	0110
$Result = B$	Move B	0111
Logic Operations		
$result = A \wedge B$	AND	1000
$result = A \vee B$	OR	1001
$result = A \oplus B$	Exclusive-OR (EOR)	1010
$result = \overline{B}$	Complement	1011
$result = \overline{B} - 1$	Decrement	1100

Alignment Unit

Maps bits in the Instruction Format to fields needed by the datapath



Register Address Logic

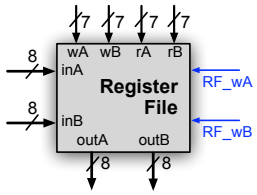


Maps opcode, A, Rd, and Rr to rA, rB, wA, and wB.

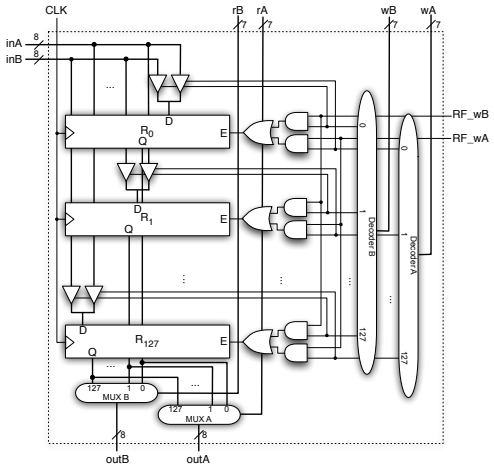
RAL automatically generates X, Y, and Z based on opcode.

Register File

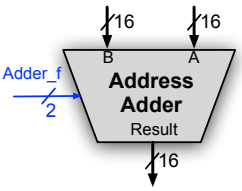
2-read-port, 2 write-port



Address Register read in one cycle (e.g., X-, Y-, or Z).



Address Adder

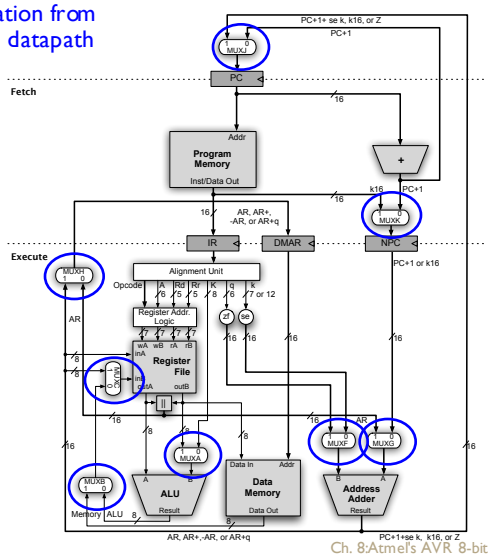


Operation	Description	Adder_f
$Result = A + B$	Add	00
$Result = A + 1$	Increment	01
$Result = A - 1$	Decrement	10
$Result = A$	Move A	11

Generates target addresses for branches and jumps.

MUXs

Routes information from one part of the datapath to another

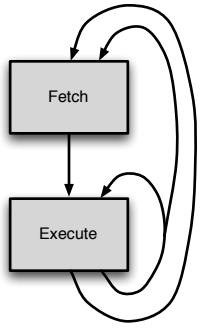


8.4 Multi-cycle Implementation

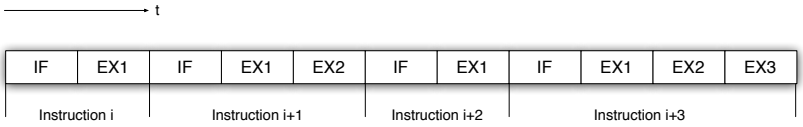
Fetch and Execute Stages

Spends one cycle in the Fetch stage to fetch an instruction.

Spends one or more cycles in the Execute and Fetch stages to execute an instruction.



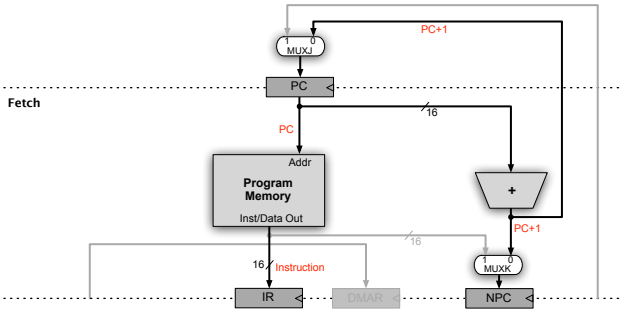
Multi-cycle Implementation



Instruction cycles (Fetch and Execute) executed sequentially.

Will see pipeline implementation later.

Fetch Cycle

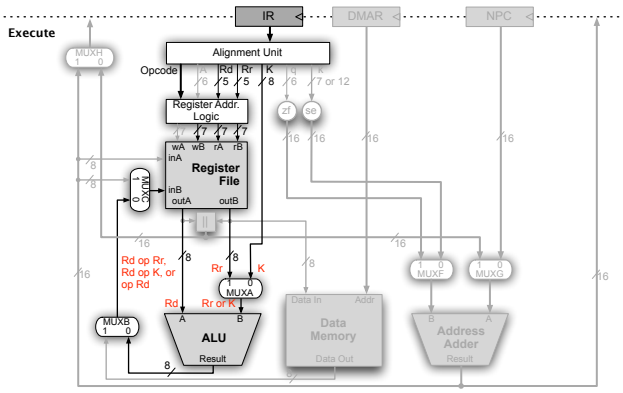


Stage	Micro-operations
IF	IR ← M[PC], PC ← PC +1, NPC ← PC +1

Execute Cycle

- Depends on the instruction
  - Arithmetic and Logic
  - Data transfer
  - Branch and jump
- Different parts of the datapath are activated.

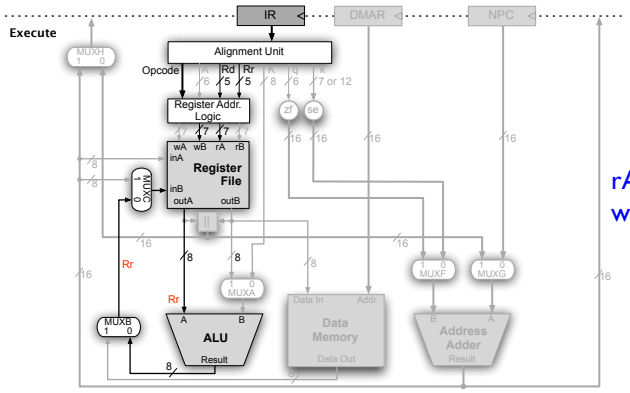
Arithmetic and Logic Instructions



Op Rd, Rr
Op Rd, K
Op Rd

Arithmetic and Logic Instructions			
Stage	Micro-operations		
	Binary	Immediate	Unary
EX	Rd ← Rd *op* Rr	Rd ← Rd *op* K	Rd ← *op* Rd

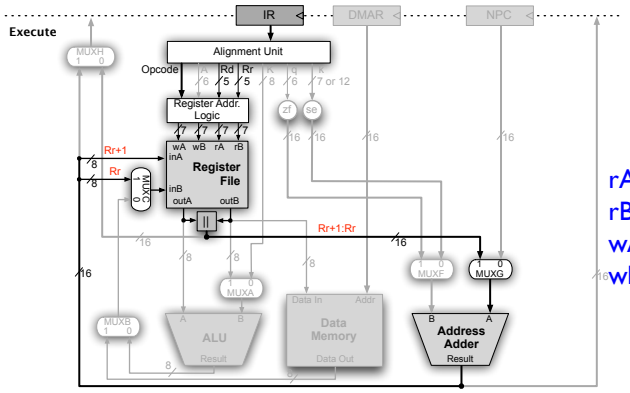
Data Transfer: MOV (8-bit)



rA = Rr
wB = Rd

Move and I/O Instructions				
Stage	Micro-operations			
	Move 8-bit	In	Out	Move 16-bit
EX	Rd ← Rr	Rd ← A	A ← Rr	Rd+1:Rd ← Rr+1:Rr

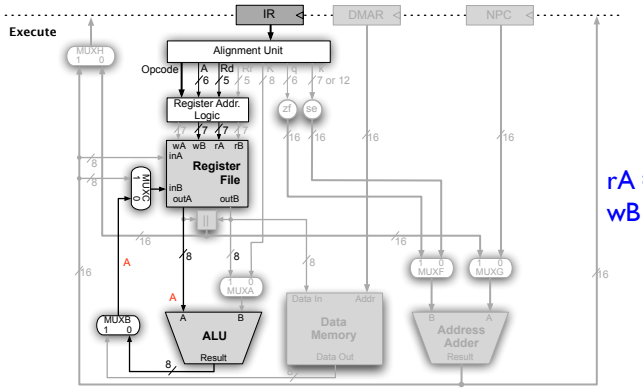
Data Transfer: MOVW (16-bit)



rA = Rr+1
rB = Rr
wA = Rd + 1
wB = Rd

Move and I/O Instructions				
Stage	Micro-operations			
	Move 8-bit	In	Out	Move 16-bit
EX	Rd ← Rr	Rd ← A	A ← Rr	Rd+1:Rd ← Rr+1:Rr

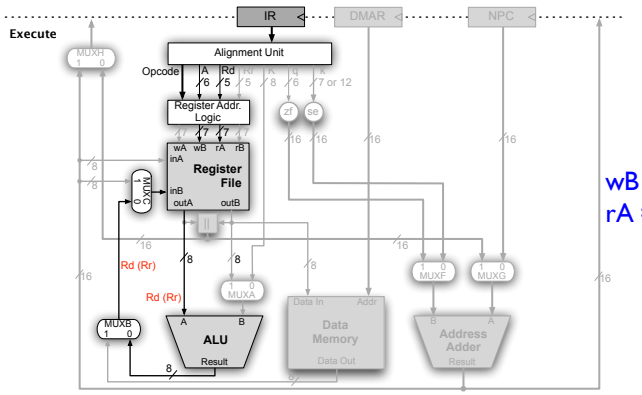
Data Transfer: IN



rA = A + 32
wB = Rd

Move and I/O Instructions				
Stage	Micro-operations			
	Move 8-bit	In	Out	Move 16-bit
EX	Rd ← Rr	Rd ← A	A ← Rr	Rd+1:Rd ← Rr+1:Rr

Data Transfer: OUT



wB = A + 32
rA = Rd (Rr)

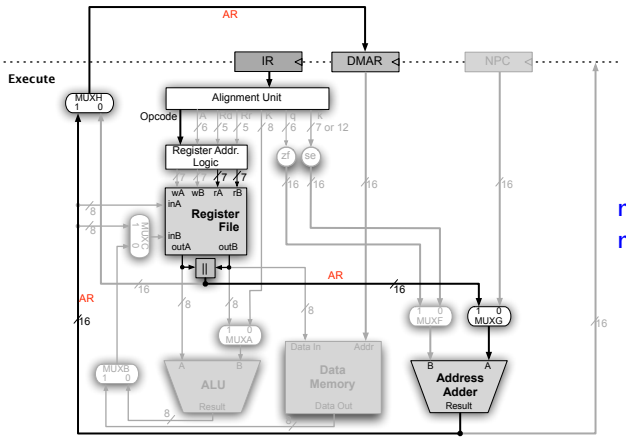
Move and I/O Instructions				
Stage	Micro-operations			
	Move 8-bit	In	Out	Move 16-bit
EX	Rd ← Rr	Rd ← A	A ← Rr	Rd+1:Rd ← Rr+1:Rr

Load and Store

- Uses Address Register (AR): X-, Y-, and Z-register.
  - Displacement
  - Pre-decrement
  - Post-increment

Load and Store Instructions				
Stage	Micro-operations			
	Normal	Displacement	Pre-Decrement	Post-Increment
EX1	DMAR ← AR	DMAR ← AR+q	DMAR ← AR-1, AR ← AR -1	DMAR ← AR, AR ← AR +1
EX2	Loads		Stores	
	Rd ← M[DMAR]		M[DMAR] ← Rr	

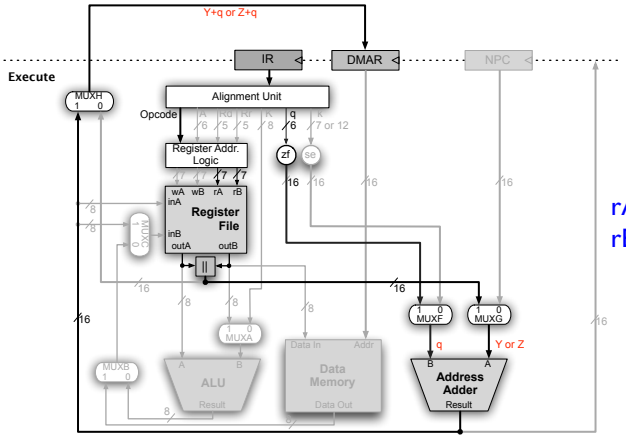
Indirect Load and Store – EX1



rA = ARh
rB = ARl

Load and Store Instructions				
Stage	Micro-operations			
	Normal	Displacement	Pre-Decrement	Post-Increment
EX1	DMAR ← AR	DMAR ← AR+q	DMAR ← AR-1, AR ← AR -1	DMAR ← AR, AR ← AR +1
EX2	Loads		Stores	
	Rd ← M[DMAR]		M[DMAR] ← Rr	

Indirect \w Disp.– EX1

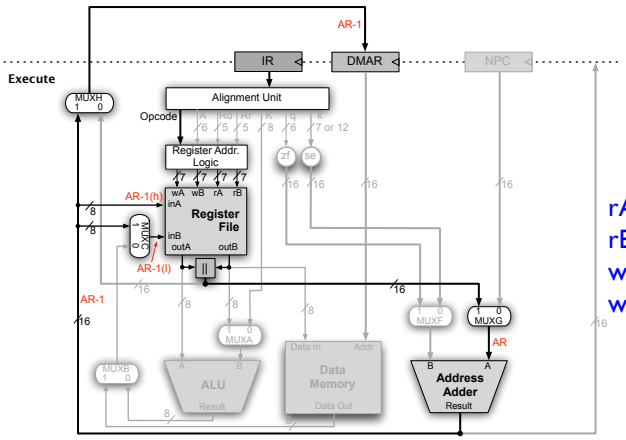


rA = ARh
rB = ARl

Load and Store Instructions				
Stage	Micro-operations			
	Normal	Displacement	Pre-Decrement	Post-Increment
EX1	DMAR ← AR	DMAR ← AR+q	DMAR ← AR-1, AR ← AR -1	DMAR ← AR, AR ← AR +1
EX2	Loads		Stores	
	Rd ← M[DMAR]		M[DMAR] ← Rr	

33

Indirect \w Pred.– EX1

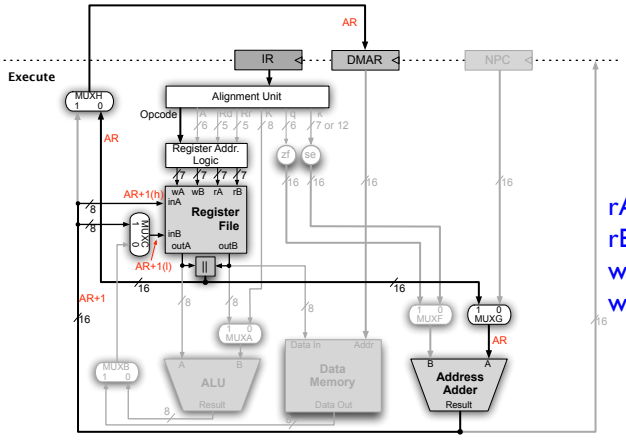


rA = ARh
rB = ARl
wA = ARh
wB = ARl

Load and Store Instructions				
Stage	Micro-operations			
	Normal	Displacement	Pre-Decrement	Post-Increment
EX1	DMAR ← AR	DMAR ← AR+q	DMAR ← AR-1, AR ← AR -1	DMAR ← AR, AR ← AR +1
EX2	Loads		Stores	
	Rd ← M[DMAR]		M[DMAR] ← Rr	

34

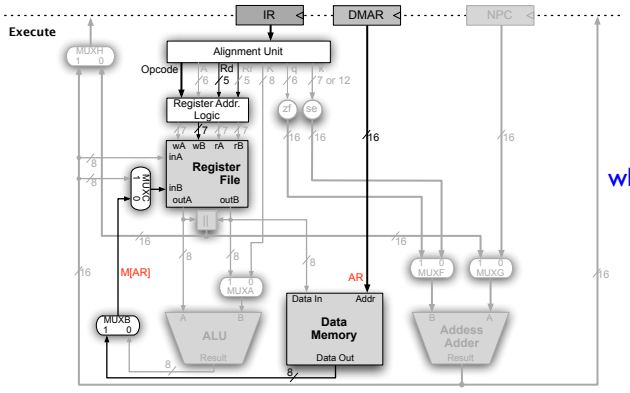
Indirect \w Post. - EX1



rA = ARh
rB = ARl
wA = ARh
wB = ARl

Load and Store Instructions				
Stage	Micro-operations			
	Normal	Displacement	Pre-Decrement	Post-Increment
EX1	DMAR ← AR	DMAR ← AR+q	DMAR ← AR-1, AR ← AR -1	DMAR ← AR, AR ← AR +1
EX2	Loads		Stores	
	Rd ← M[DMAR]		M[DMAR] ← Rr	

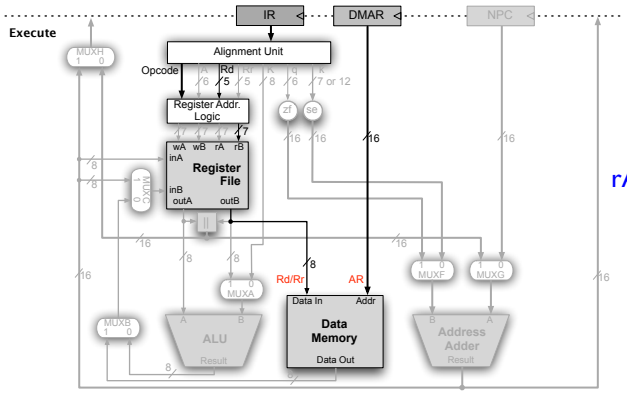
Load – EX2



wB = Rd

Load and Store Instructions				
Stage	Micro-operations			
	Normal	Displacement	Pre-Decrement	Post-Increment
EX1	DMAR ← AR	DMAR ← AR+q	DMAR ← AR-1, AR ← AR -1	DMAR ← AR, AR ← AR +1
EX2	Loads		Stores	
	Rd ← M[DMAR]		M[DMAR] ← Rr	

Store – EX2

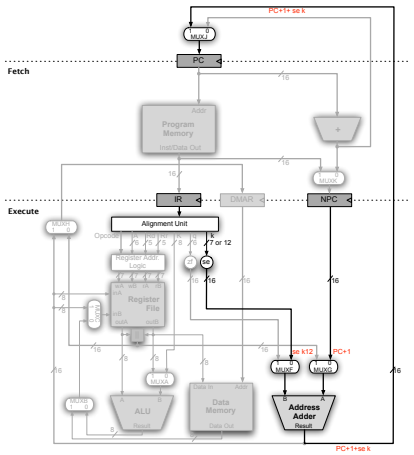


rA = Rd (Rr)

Load and Store Instructions				
Stage	Micro-operations			
	Normal	Displacement	Pre-Decrement	Post-Increment
EX1	DMAR ← AR	DMAR ← AR+q	DMAR ← AR-1, AR ← AR -1	DMAR ← AR, AR ← AR +1
EX2	Loads		Stores	
	Rd ← M[DMAR]		M[DMAR] ← Rr	

37

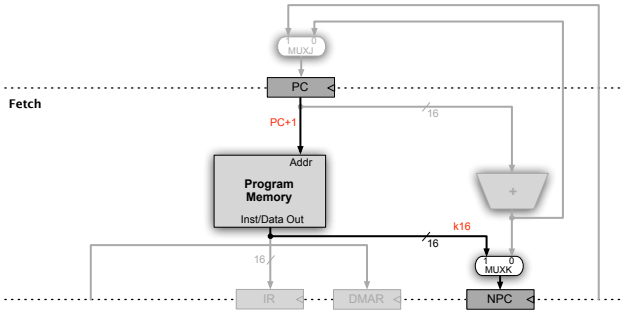
PC-relative Branch – EX1



Branch and Jump Instructions			
Stage	Micro-operations		
	PC-relative	Direct	Indirect
EX1	If (*flag*) then PC←NPC + se k	NPC ← M[PC]	PC ← Z
EX2		PC ← NPC	

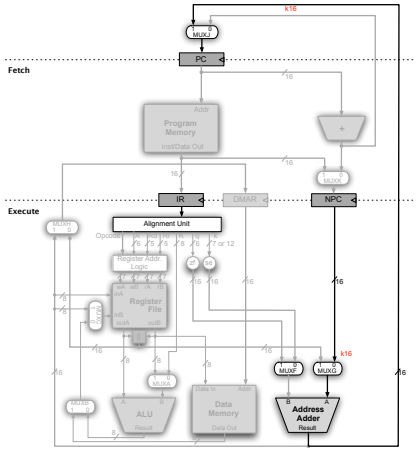
38

Jump Direct – EX1



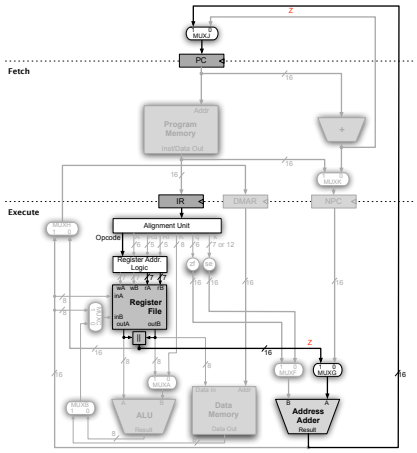
Branch and Jump Instructions			
Stage	Micro-operations		
	PC-relative	Direct	Indirect
EX1	If (*flag*) then PC←NPC + se k	NPC ← M[PC]	PC ← Z
EX2		PC ← NPC	

Jump Direct – EX2



Branch and Jump Instructions			
Stage	Micro-operations		
	PC-relative	Direct	Indirect
EX1	If (*flag*) then PC←NPC + se k	NPC ← M[PC]	PC ← Z
EX2		PC ← NPC	

Jump Indirect – EX1



Branch and Jump Instructions			
Stage	Micro-operations		
	PC-relative	Direct	Indirect
EX1	If (*flag*) then PC←NPC + se k	NPC ← M[PC]	PC ← Z
EX2		PC ← NPC	

8.5 Execution of More Complex Instructions

Enhanced AVR Datapath

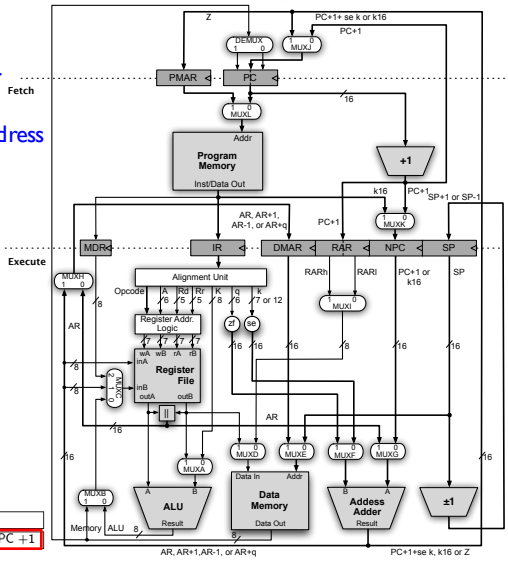
Additional Registers:
RAR – Return Address Register
MDR – Memory Data Register
PMAR – Program Memory Address Register

Concatenation
PC = PCh || PCl
RAR = RARh || RARl

Additional MUXs:
MUXD
MUXE
MUXI
DEMUX

SP Manipulation:
Increment/Decrement Unit

Stage	Micro-operations
IF	IR ← M[PC], PC ← PC +1, NPC ← PC +1, RAR ← PC +1



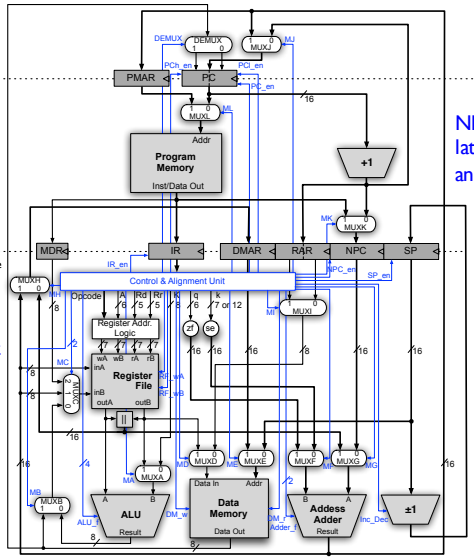
Control Signals

3 ways to latch PC
PC: PC_en
PCh: PCh_en
PCl: PCl_en
Only one can be enabled at a time.

CAU => Control Unit + Alignment Unit

NPC_en enables latching of both NPC and RAR

Operation	Description	Inc_Dec
SP+1	Increment	0
SP-1	Decrement	1



8.6 Control Unit Design

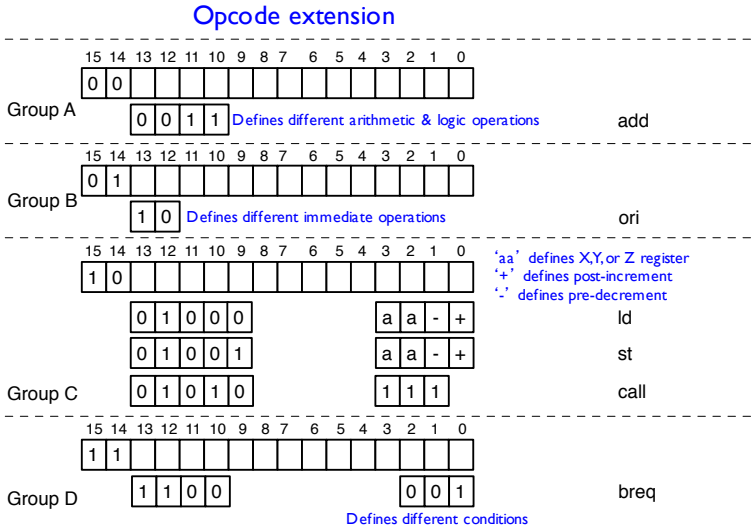
Instructions Considered

Representative AVR Instructions for Control Unit Design

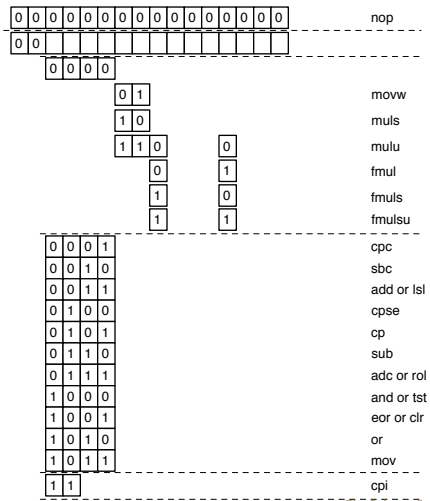
Category	Mnemonics	Description	Operation	Flags	# of Execute cycles
ALU	ADD Rd, Rr	Add two Registers	Rd← Rd + Rr	Z,C,N,V,H	1
	ORI Rd, K	Logical OR Register and Constant	Rd← Rd ∨ K	Z,C,N,V,H	1
Data Transfer	LD Rd, Y	Load Indirect	Rd ← M[Y]	None	2
	ST Y, Rr	Store Indirect	M[Y] ← Rr	None	2
Branch	BREQ k	Branch if Equal	if (Z=1) then PC←PC+1+k	None	2/3
	CALL k	Direct Subroutine call	PC←k, STACK←PC+1	None	

Defined in "Atmel 8-bit AVR Instruction Set"

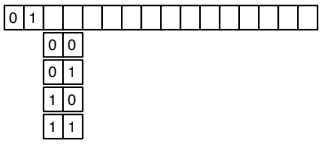
Opcode Encoding



Opcode Encoding – Group A

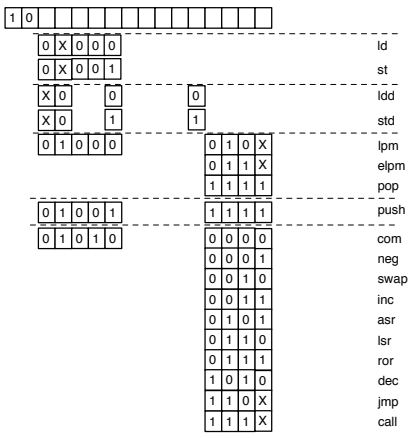


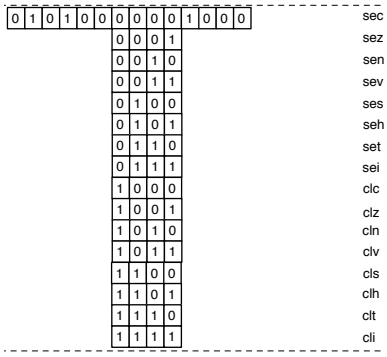
Opcode Encoding – Group B



sbci
subi
ori or sbr
andi or cbr

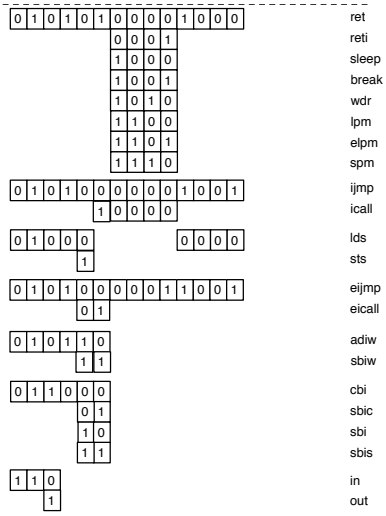
Opcode Encoding – Group C



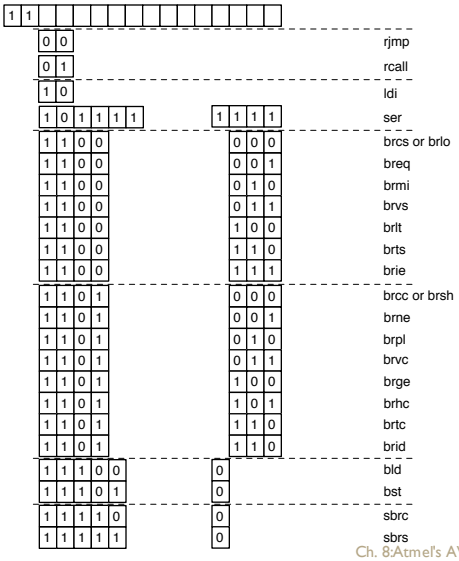


Group C encodes the largest number of instructions

Opcode Encoding – Group C

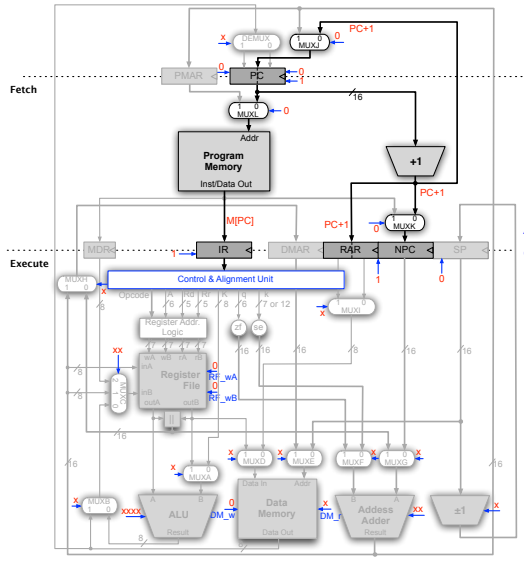


Opcode Encoding – Group D



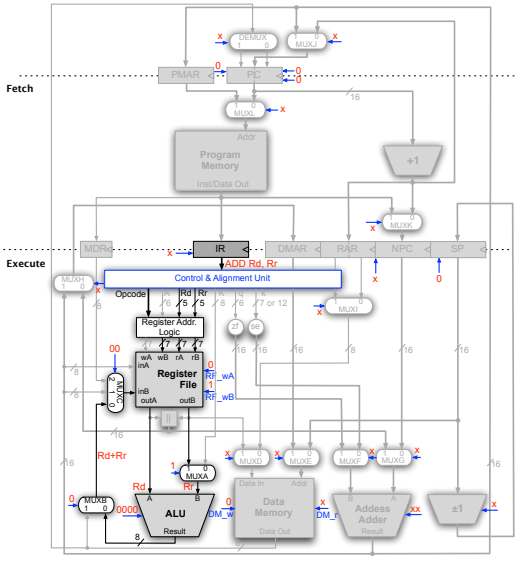
Fetch Stage

Stage	Micro-operations
IF	IR ← M[PC], PC ← PC +1, NPC ← PC +1, RAR ← PC +1



Also latches PC+1 onto RAR

ADD – EX1



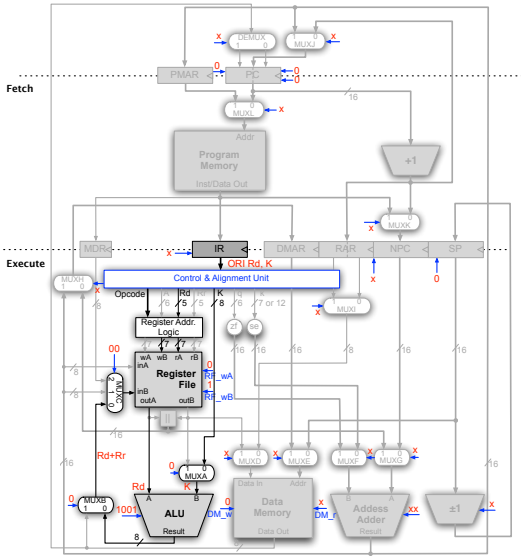
PC_en must be 0 so that it points to the next instruction be fetched.

IR_en in the last EX cycle can be "don't care", since Fetch stage will fetch a new instruction.

Data Memory must not be modified.

SP must not be modified.

ORI – EX1



PC_en must be 0 so that it points to the next instruction be fetched.

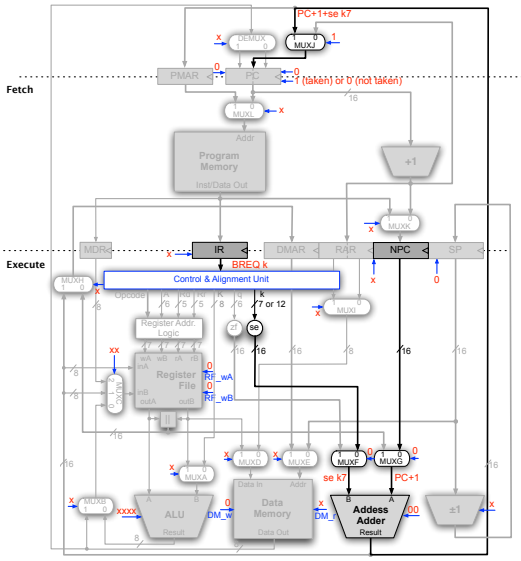
IR_en in the last EX cycle can be "don't care", since Fetch stage will fetch a new instruction.

Data Memory must not be modified.

SP must not be modified.

55

BREQ – EX1



PC_en needs to be 0 or 1 depending Z-flag.

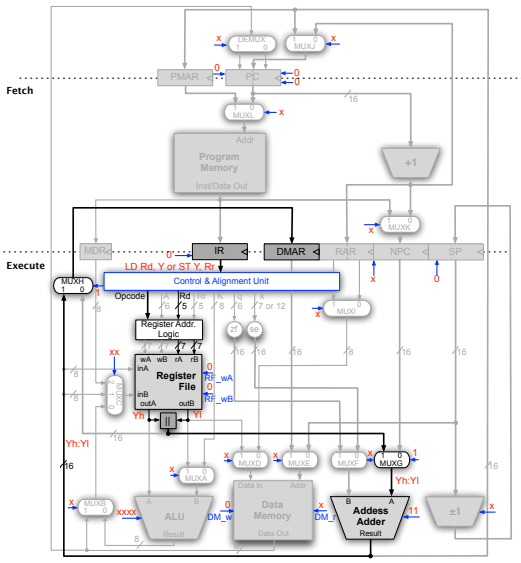
IR_en in the last EX cycle can be "don't care", since Fetch stage will fetch a new instruction.

Data Memory must not be modified.

SP must not be modified.

56

LD and ST – EX1



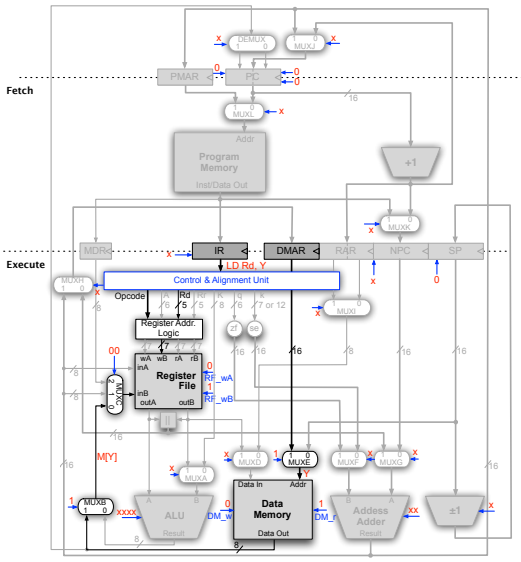
PC_en must be 0 so that it points to the next instruction be fetched.

IR_en must be 0 to provide instruction to the subsequent cycle.

Data Memory must not be modified.

SP must not be modified.

LD – EX2

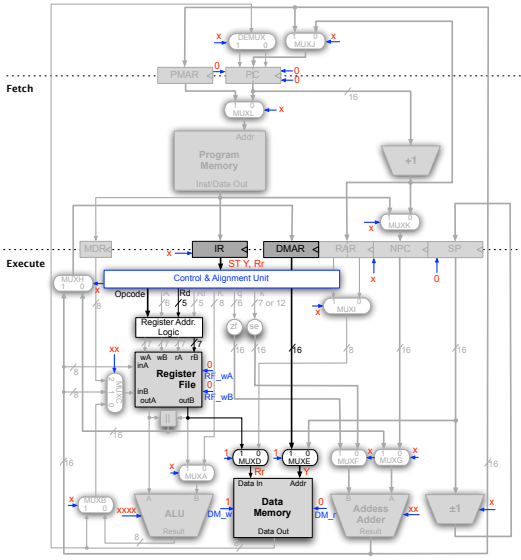


PC_en must be 0 so that it points to the next instruction be fetched.

IR_en in the last EX cycle can be "don't care", since Fetch stage will fetch a new instruction.

SP must not be modified.

ST – EX2

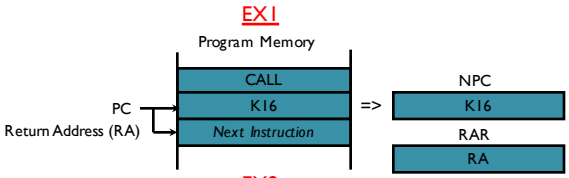


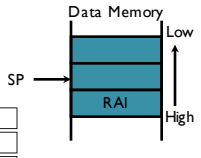
PC_en must be 0 so that it points to the next instruction be fetched.

IR_en in the last EX cycle can be "don't care", since Fetch stage will fetch a new instruction.

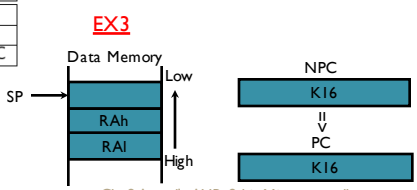
SP must not be modified.

Direct Subroutine Call

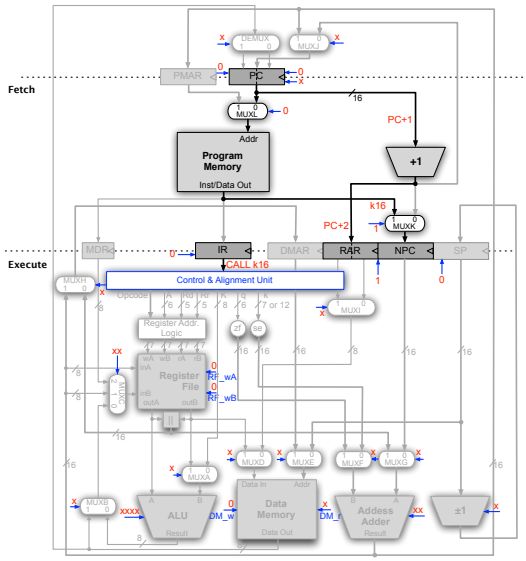




Direct Subroutine Call	
Stage	Micro-operations
EX1	NPC ← M[PC], RAR ← PC + 1
EX2	M[SP] ← RARl, SP ← SP -1
EX3	M[SP] ← RARh, SP ← SP -1, PC ← NPC



CALL – EX1



PC_en can be "don't care" since it will be overwritten in EX3.

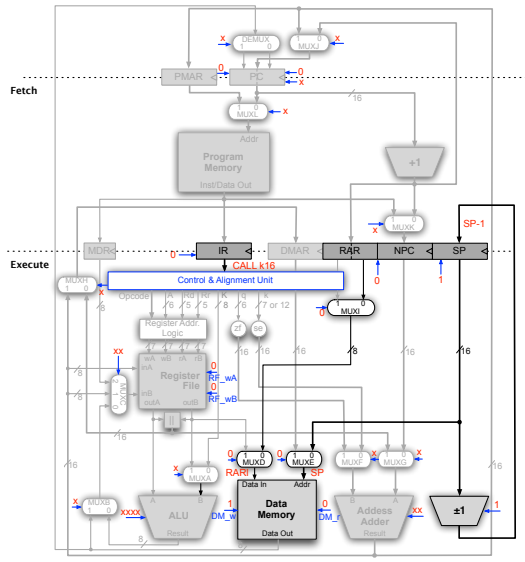
IR_en must be 0 to provide instruction to the subsequent cycle.

SP must not be modified.

Subroutine return address is latched onto RAR.

61

CALL – EX2



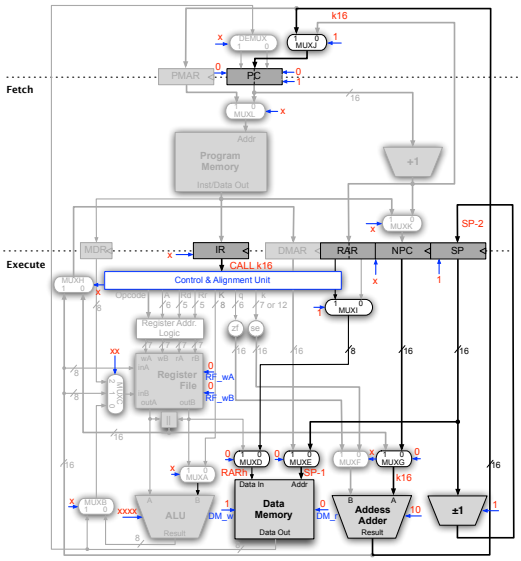
PC_en can be "don't care" since it will be overwritten in EX3.

IR_en must be 0 to provide instruction to the subsequent cycle.

RAR must not be modified.

62

CALL – EX3



PC updated.

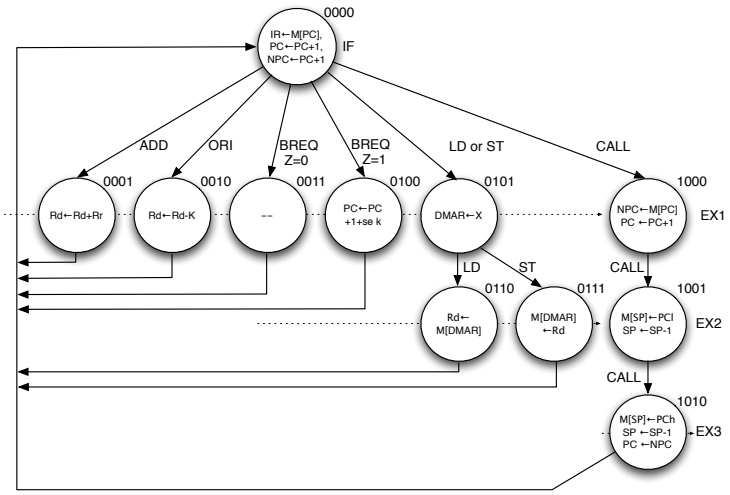
IR_en in the last EX cycle can be "don't care", since Fetch stage will fetch a new instruction.

Summary of Control Signals

Control Signals	IP	Instructions										
		ADD	ORI	BREQ		LD		ST		CALL		
		EX1	EX1	EX1		EX1	EX2	EX1	EX2	EX1	EX2	EX3
				Z=0	Z=1							
MJ	0	x	x	1	1	x	x	x	x	x	x	1
MK	0	x	x	x	x	x	x	x	x	1	x	x
ML	0	x	x	x	x	x	x	x	x	0	x	x
IR_en	1	x	x	x	x	0	x	0	x	0	0	x
PC_en	1	0	0	0	1	0	0	0	0	x	x	1
PCh_en	0	0	0	0	0	0	0	0	0	0	0	0
PCl_en	0	0	0	0	0	0	0	0	0	0	0	0
NPC_en	1	x	x	x	x	x	x	x	x	1	0	x
SP_en	0	0	0	0	0	0	0	0	0	0	1	1
DEMUX	x	x	x	x	x	x	x	x	x	x	x	x
MA	x	1	0	x	x	x	x	x	x	x	x	x
MB	x	0	0	x	x	x	1	x	x	x	x	x
ALU_f	xxxx	0000	1001	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
MC	xx	00	00	xx	xx	xx	00	xx	xx	xx	xx	xx
RF_wA	0	0	0	0	0	0	0	0	0	0	0	0
RF_wB	0	1	1	0	0	0	1	0	0	0	0	0
MD	x	x	x	x	x	x	x	x	1	x	0	0
ME	x	x	x	x	x	x	1	x	1	x	0	0
DM_r	x	x	x	x	x	x	1	x	0	x	0	0
DM_w	0	0	0	0	0	0	0	0	1	0	1	1
MF	x	x	x	0	0	x	x	x	x	x	x	x
MG	x	x	x	0	0	1	x	1	x	x	x	0
Adder_f	xx	xx	xx	00	00	11	xx	11	xx	xx	xx	10
Inc_Dec	x	x	x	x	x	x	x	x	x	x	1	1
MH	x	x	x	x	x	1	x	1	x	x	x	x
MI	x	x	x	x	x	x	x	x	x	x	0	1

8.7 Finite State Implementation of the Control Unit

Sequence Control

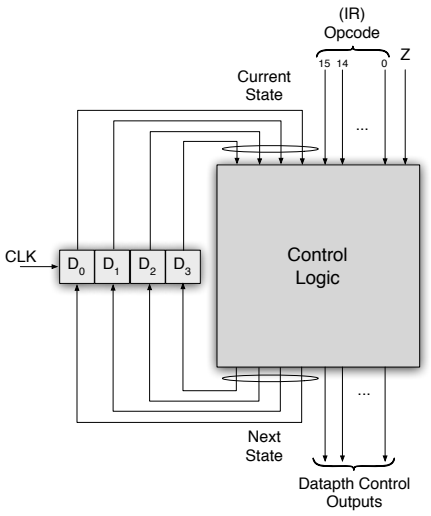


Finite State Table

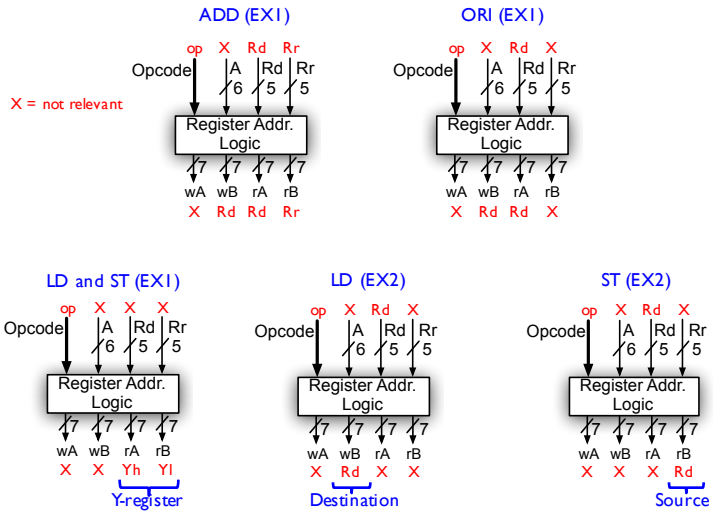
Description	Current State	Inputs: Opcode $b_{15}b_{14}b_{10}b_9$	Inputs: Z	Next State	Outputs: MJ	MK	ML	IR_en	PC_en	PCh_en	PCL_en	MPC_en	DEMUX	MA	MB	ALU_f	MC	RF_wA	RF_wB	MD	ME	DM_r	DM_w	MF	MG	Adder_f	Inc_Dec	MH	MI
IF	0000	xxxx	x	–	0	0	0	1	1	0	0	1	x	x	x	xxxx	xx	0	0	x	x	x	0	x	x	xx	x	x	x
ADD	0001	00xx	x	0000	x	x	x	x	0	0	0	x	x	1	0	0000	00	0	1	x	x	x	0	x	x	xx	x	x	x
ORI	0010	01xx	x	0000	x	x	x	x	0	0	0	x	x	0	0	1001	00	0	1	x	x	x	0	x	x	xx	x	x	x
BREQ (Z=0)	0011	11xx	0	0000	1	x	x	x	0	0	0	x	x	x	x	xxxx	xx	0	0	x	x	x	0	0	0	00	x	x	x
BREQ (Z=1)	0100	11xx	1	0000	1	x	x	x	1	0	0	x	x	x	x	xxxx	xx	0	0	x	x	x	0	0	0	00	x	x	x
LD/ST (EX1)	0101	100x	x	–	x	x	x	0	0	0	0	x	x	x	x	xxxx	xx	0	0	x	x	x	0	x	1	11	x	1	x
LD (EX2)	0110	1000	x	0000	x	x	x	x	0	0	0	x	x	x	1	xxxx	00	0	1	x	1	1	0	x	x	xx	x	x	x
ST (EX2)	0111	1001	x	0000	x	x	x	x	0	0	0	x	x	x	x	xxxx	xx	0	0	1	1	0	1	x	x	xx	x	x	x
CALL (EX1)	1000	1010	x	1001	x	1	0	0	x	0	0	1	x	x	x	xxxx	xx	0	0	x	x	x	0	x	x	xx	x	x	x
CALL (EX2)	1001	xxxx	x	1010	x	x	x	0	x	0	0	0	x	x	x	xxxx	xx	0	0	0	0	0	1	x	x	xx	x	x	0
CALL (EX3)	1010	xxxx	x	0000	1	x	x	x	1	0	0	x	x	x	x	xxxx	xx	0	0	0	0	0	1	x	0	10	x	x	1

- IF is independent of inputs
- State transition from 0000 (IF) depends on opcode
- State transition from 0000 (IF) to 0011 (BREQ, Z =0) or 0100 (BREQ, Z=1) depends on opcode and Z.
- State transition from 0101 (LD/ST, EX1) to 0110 (LD, EX2) or 0111 (ST, EX2) depends only on bit 9 of opcode.
- State transition from 1000 to 1001 and from 1001 to 1010 depends only on the current state.

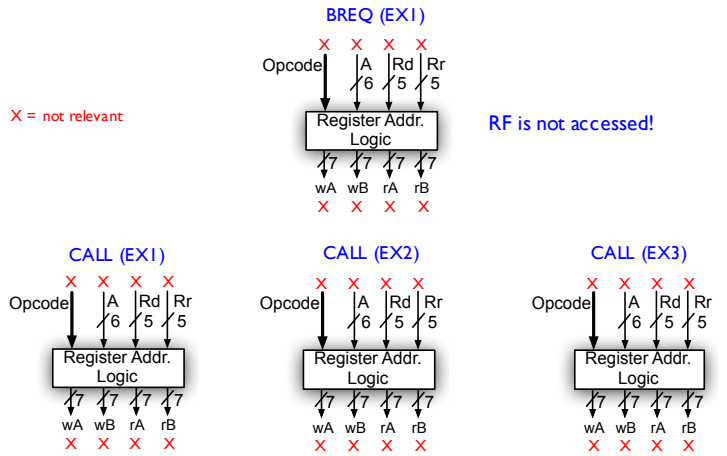
FSM



RAL Mapping



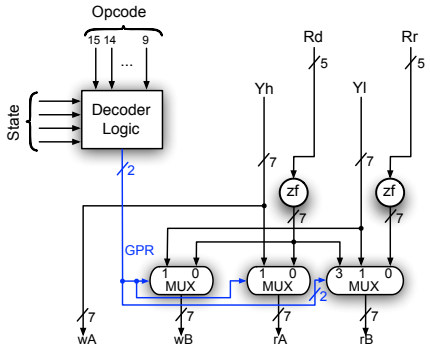
RAL Mapping (cont.)



RF is not accessed!
Separate Inc/Dec Unit and path for manipulating SP

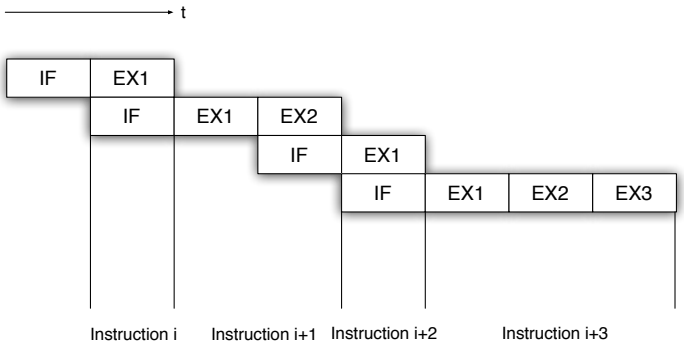
Register Address Logic

RF R/W Ports	IF	Instructions									
		ADD	ORI	BREQ	LD		ST		CALL		
		EX1	EX1	EX1	EX1	EX2	EX1	EX2	EX1	EX2	EX3
wA	x	x	x	x	x	x	x	x	x	x	x
wB	x	Rd	Rd	x	x	Rd	x	x	x	x	x
rA	x	Rd	Rd	x	Yh	x	Yh	x	x	x	x
rB	x	Rr	Rr	x	Yl	x	Yl	Rd	x	x	x

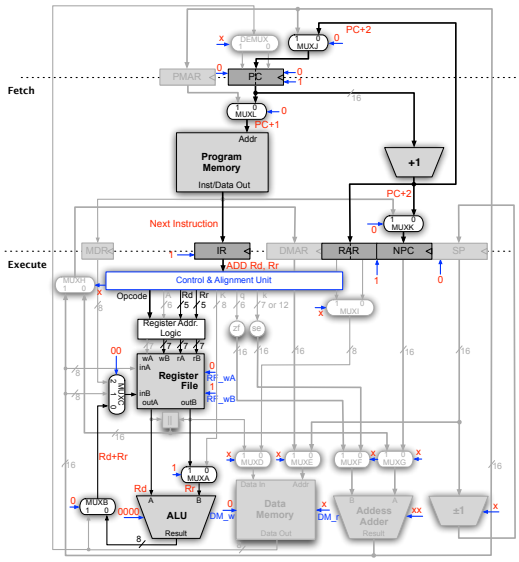


8.8 Pipeline Implementation

Pipeline Implementation

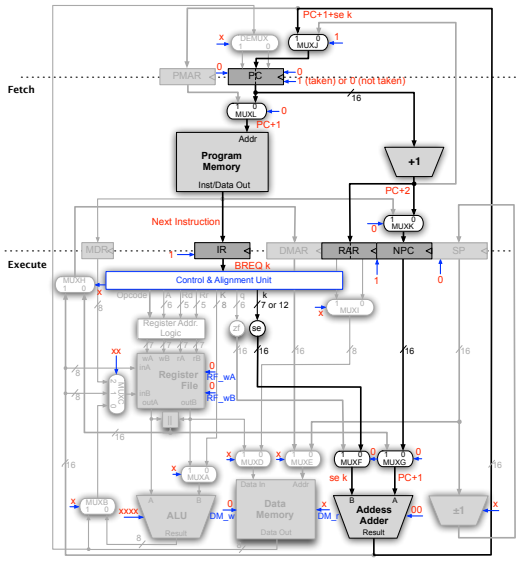


Execution of ADD and Fetch



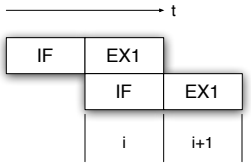
Execution of ORI and Fetch is similar.

Execution of BREQ and Fetch

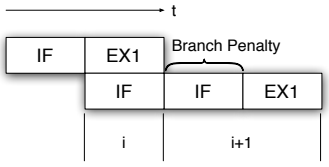


Execution of any conditional branch instruction and Fetch is similar.

Execution of BREQ and Fetch

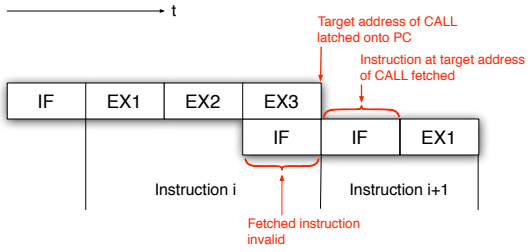


BREQ not taken

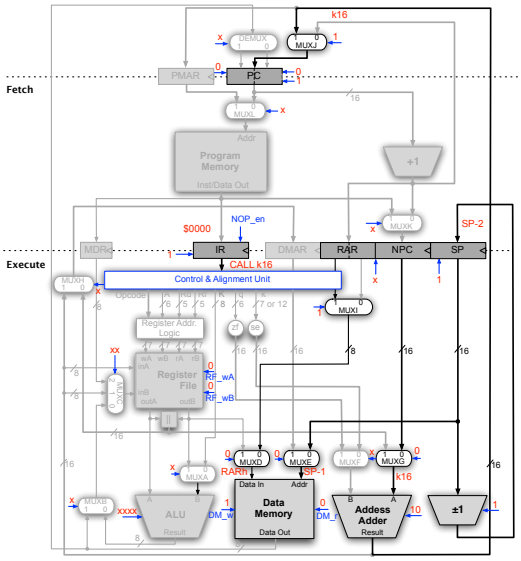


BREQ taken

Execution of CALL-EX3 & Fetch



Execution of CALL-EX3 & Fetch



Creates a bubble by executing a NOP instruction - $0000.

Questions?