Disclaimer

The slides presented here are a combination of the CS251 course notes from previous terms, the work of Xiao-Bo Li, and material from the required textbook "Computer Organization and Design, ARM Edition," by David A. Patterson and John L. Hennessy. It is being used here with explicit permission from the authors.

CS251 course policy requires students to delete all course files after the term. Therefore, please do not post these slides to any website or share them.

CS251 - Computer Organization and Design

Single Cycle Processor Implementation

Instructor: Zille Huma Kamal

University of Waterloo

Spring 2023

Single-Cycle Processor Implementation

- How to build datapath, control for specific architecture
- We will implement small subset of ARM operations:
  - Load (`LDUR`) and store (`STUR`)
  - Add (`ADD`), subtract (`SUB`), and (`AND`), or (`ORR`).
  - Compare and branch on zero (`CBZ`) and branch (`B`)
- These suffice to illustrate fundamental ideas

Review of ARM Architecture

- 32 registers (numbered 0 to 31), each with 64 bits
- Register 31 (X31 or XZR) always supplies the value 0
- Data Memory has 64-bit words (double-words)
- Instruction memory has 32 bits words
- Memory is byte-addressable
  (word addresses multiples of 4, double-word addresses multiples of 8)

Think About It

Words have multiple bytes, which byte should be the address of the word?

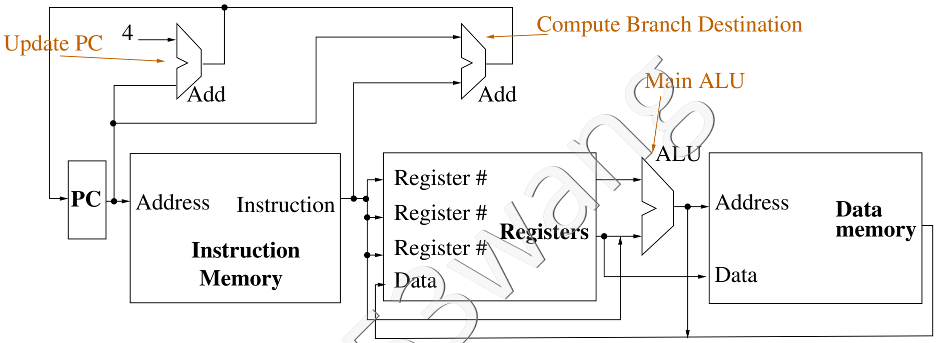
- (Double-)Words have the address of their most significant byte[a]

[a] in ARMv8 this is dynamic, x86 prefers LSB as word address

Review of ARM Instructions

- Load: `LDUR X1, [X2,#200]`
  - Operands are register to be loaded, address in memory
  - Addressing modes: register, base (displacement), immediate
- Add: `ADD X1, X2, X3`
  - Operands are destination register, two source registers
- Conditional branch: `CBZ X1, #10`
  - Operand is registers to compare to 0, relative jump offset
  - Addressing: PC relative
- Branch: `B 3000`
  - Operand is word offset of next instruction
  - Addressing mode: PC-relative
    (need to multiply by 4)

High-Level View of ARM Functional Units

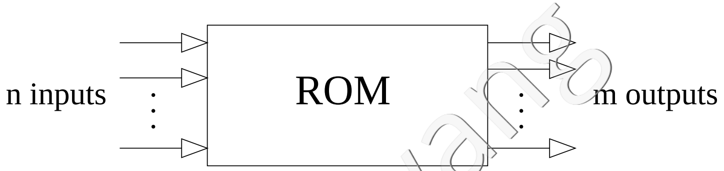


- PC: Program Counter (address of current instruction)
- Fetch-execute cycle:
  - Fetch instruction (update PC)
  - Execute instruction
    - Fetch register operands
    - Compute result
    - Store into registers OR use to index memory

Storage/State Elements

- In the high-level view of ARM functional units. We see two main storage elements: Registers and Memory (Instruction and Data).
- We will discuss the following **primary** storage/state elements:
  - ROM
  - Registers
  - SRAM
  - DRAM
- **Secondary** storage includes magnetic hard disk and flash disk drives, which we will discuss later in the course

Implementing Boolean Functions: ROMs

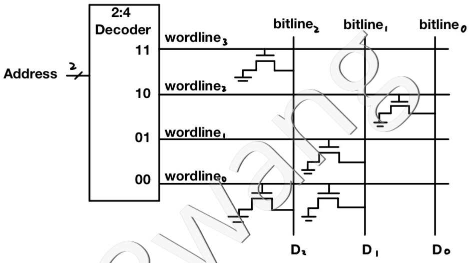


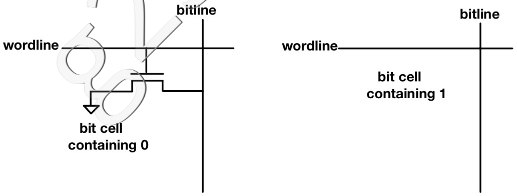
- Can think of ROM as table of 2^n m-bit words
- Can think of ROM as implementing m one-bit functions of n variables
- Internally, consists of a decoder plus an OR gate for each output
- Types of ROM: PROM, EPROM, EEPROM
- PLAs - simplified ROM
  Less hardware, but less flexible

Example: ROM

Here is an example of a ROM with 2^2 3-bit words.

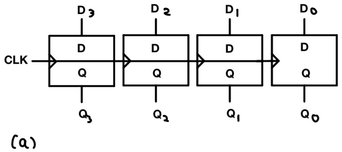
A_1	A_0	D_2	D_1	D_0
1	1	0	1	1
1	0	1	1	0
0	1	1	0	1
0	0	0	0	1

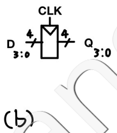




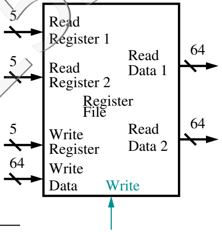
Registers and Register Files

- Register: an array of flip-flops[1] (64 for a double word register)



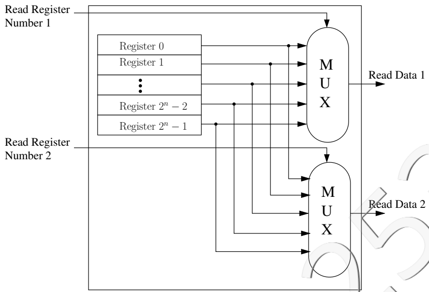


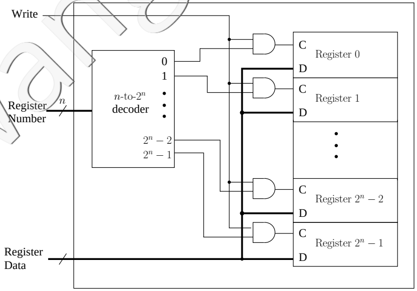
- shared clock - allows all input (all bits) of the register to be updated at the same time
- Register file: a way of organizing registers



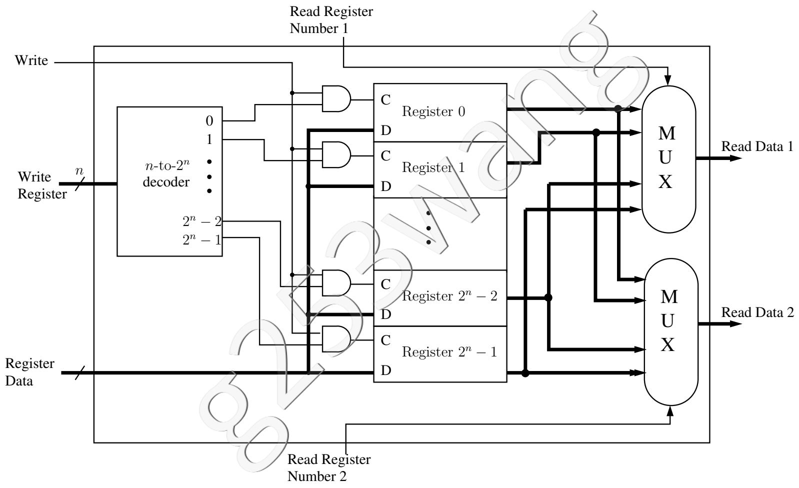
[1]only for this course, registers are actually implemented as SRAM cells

Read/Write Logic for Register File



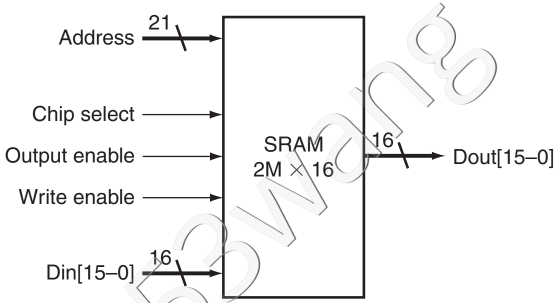


Read/Write Logic for Register File–Merged



Random Access Memories

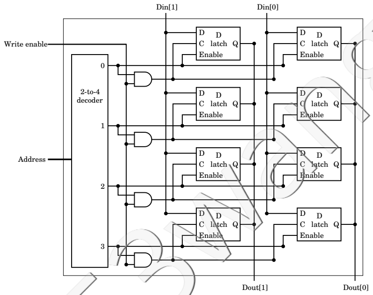
- Static random access memories (SRAM) use D latches



Similar to Figure A.9.1 from the text

- Register file idea won't scale up; decoder and multiplexors too big
- Fix multiplexor problem by using three-state buffers
- Fix decoder problem by using two-level decoding
- This type of memory is **not** clocked

Example of SRAM Structure [2]



Simlar to Figure A.9.3 from the text

Think About It

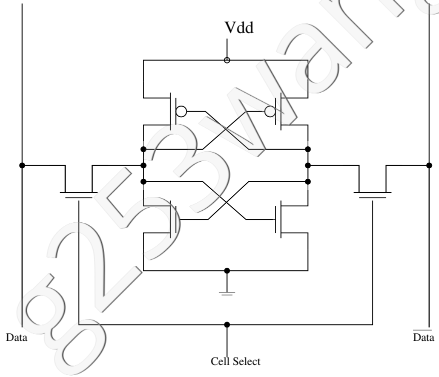
Does this design scale up well?

- No - transistor count in decoder is high

[2] the D-latch looks different from our earlier course notes. Concept remains same

6 Transistor SRAM Cell

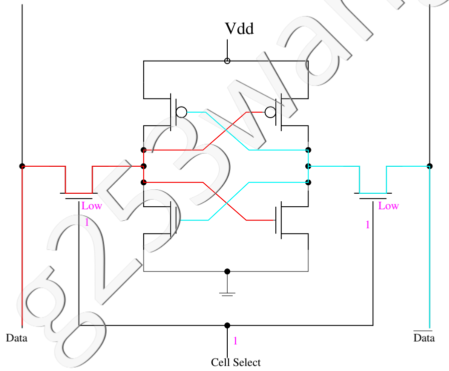
Flipflop[3] takes about 40 transistors.
Can implement SRAM cell with 6 transistors:



[3]D Latch will take less but 6T is much less

6 Transistor SRAM Cell–How It Works

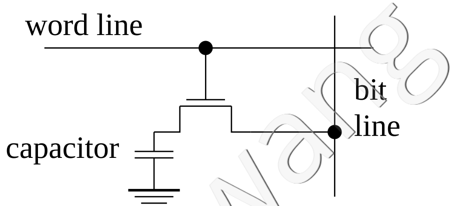
Put 1 on Cell Select, and then (for example) put 1 on Data and 0 on $\overline{Data}$



Dynamic RAM

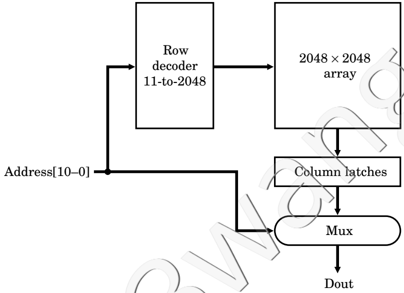
- Even six transistors is too expensive
- Alternative: use a capacitor to store a charge to represent 1
- Problem: charge leaks away, must be refreshed

DRAM Cell



- To write: place value on bit line, 1 on word line
  Change word line to 0 before changing bit line
- To read: put half-voltage on bit line, 1 on word line
  Charge in capacitor will slightly increase bit line voltage,
  no charge will slightly decrease voltage
  This change detected, amplified, and written back

Design of 4Mx1 DRAM



Similar to Figure A.9.6 from the text

- 20-bit address provided 11 bits at a time
- Whole row is read at once
- Column address selects single bit
- Refresh handled a row at a time (external controller)
- If capacitors hold charge for 4ms, refresh takes 80ns, fraction of time devoted to refresh is about 4%

DRAM Complications

- DRAM is cheaper than SRAM, but slower
- Refresh controller must also allow read/write access
- Possibility of getting more bits out at a time (e.g. page-mode RAM)
- SDRAM: synchronized DRAM
    - Uses external clock to synchronize with processor
    - Useful in memory hierarchies

Recall: High-Level View of ARM Functional Units

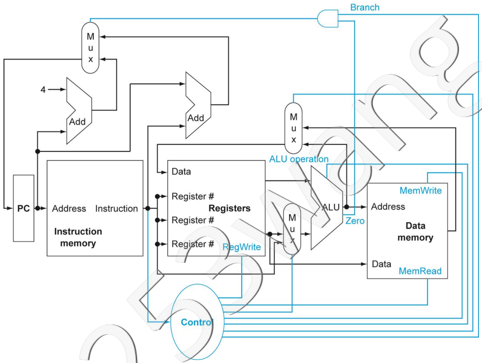


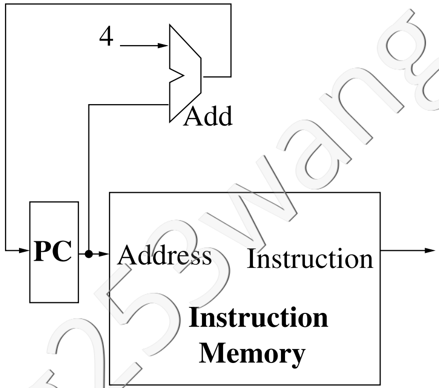
Figure 4.2 from the text

- Multiplexors used to send two signals to one location
- Control unit tells other units what to do

First Implementation: One Cycle Per Instruction

- Simpler to understand, but not practical
- Requires separate instruction and data memories
- Clock must be slowed to speed of slowest instruction
- Subsequently we look at multicycle implementations

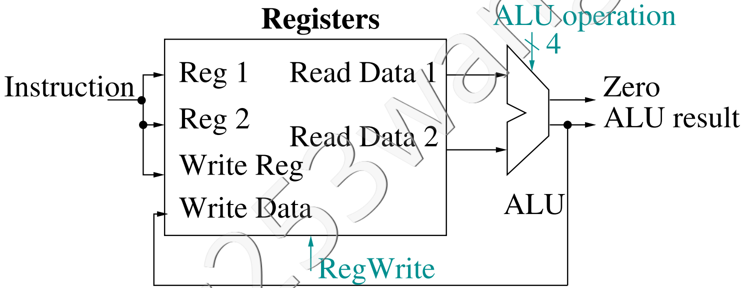
Implementing Fetch Portion of Fetch-Execute



- State elements here are PC (register) and instruction memory
- Adder is combinational

Datapath components for R-type instructions

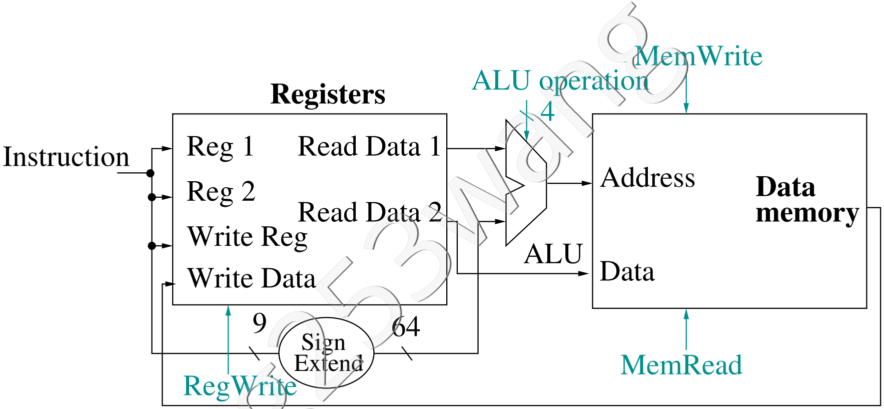
- Example: `ADD X1, X2, X3`



- Note design permits read/write of same register

Datapath components for load/store instructions

- Example: `LDUR X1, [X2,#200]`



- Sign extend is combinational
- Assume for simplicity that data memory is edge-triggered

Combined R-format, Memory

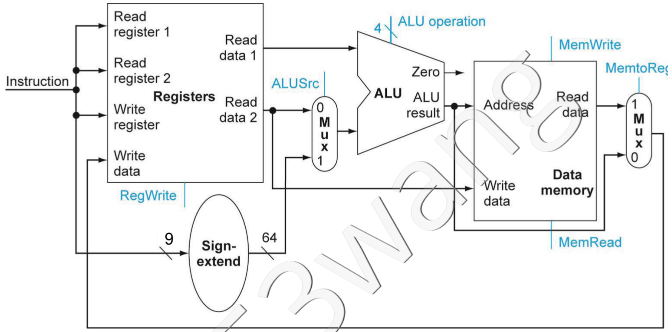
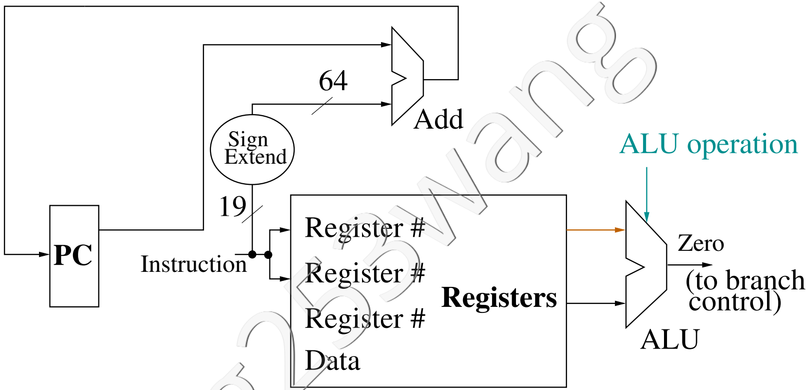


Figure 4.10 from the text

- R-format and Memory instructions are similar
- MUX in front of second ALU input selects Reg File output OR Sign-extended constant
- MUX "in front of" Reg File write data selects ALU output OR Memory Read output

Datapath components for branch instructions

- Example: `CBZ X1, #100`



- Shift is necessary because offset given is in words
- Still need mechanism to control PC loading
- Special ALUop to pass use 'b' input to ouput ('a' ignored).

Assembled Single-Cycle Datapath

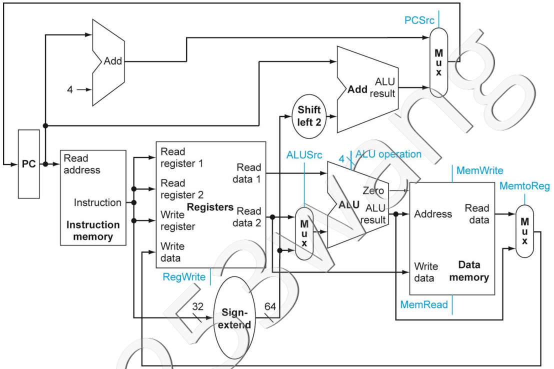
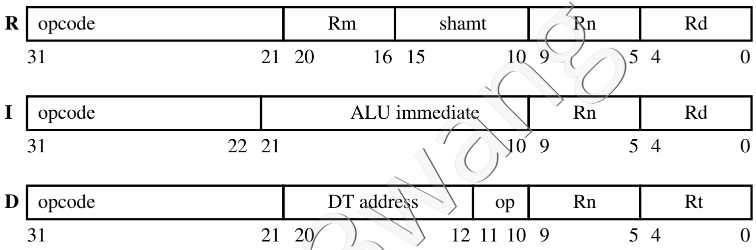


Figure 4.11 from the text

- MUX to PC
  Select between PC+4 OR CBZ address
- Note the "32-bit" input to Sign-extend

Instruction Formats



R-format: `ADD X1, X2, X3` ⇒ `ADD` **`Rd`**`,Rn,Rm`

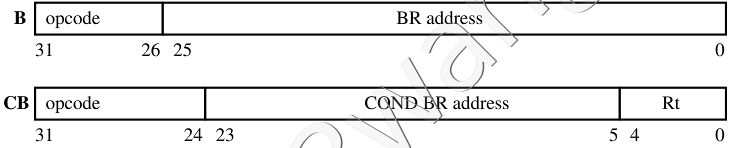
I-format: `ADDI X1, X2, #4` ⇒ `ADDI` **`Rd`**`,Rn,#4`

Load/store: `LDUR X1, [X2,#200]` ⇒ `LDUR` **`Rt`**`,[Rn,#200]`

`STUR X1, [X2,#400]` ⇒ `STUR Rt,[Rn,#400]`

(bold face register is written to)

Branch Instruction Formats



Branch: `B #3000` ⇒ `B #3000`

Conditional Branch: `CBZ X1,#3000` ⇒ `CBZ Rt,#3000`

Opcodes

Instruction	Opcode	Format
B	0001 01	B-format
ADD	1000 1011 000	R-format
ADDI	1001 0001 00	I-format
CBZ	1011 0100	CB-format
CBNZ	1011 0101	CB-format
SUB	1100 1011 000	R-format
SUBI	1101 0001 00	I-format
STUR	1111 1000 000	D-format
LDUR	1111 1000 010	D-format

- Opcodes length based on format

R-format: 11 bits	I-format: 10 bits
D-format: 11 bits	B-format: 6 bits
CB-format: 8 bits	

Meaning of Signals in Single-Cycle Datapath

Signal	Signal=0	Signal=1
Reg2Loc	20–16	4–0
Branch*	no branch	Branch
MemRead	no effect	memory read
MemToReg	reg write from ALU	reg write from memory
MemWrite	no effect	memory written
ALUSrc	ALU B input from reg	immediate from instruction
RegWrite	no effect	register written

* Branch is ANDed with Zero from ALU to get PCsrc
(Full version in Figure 4.16 of text.)

ADD X1, X2, X3

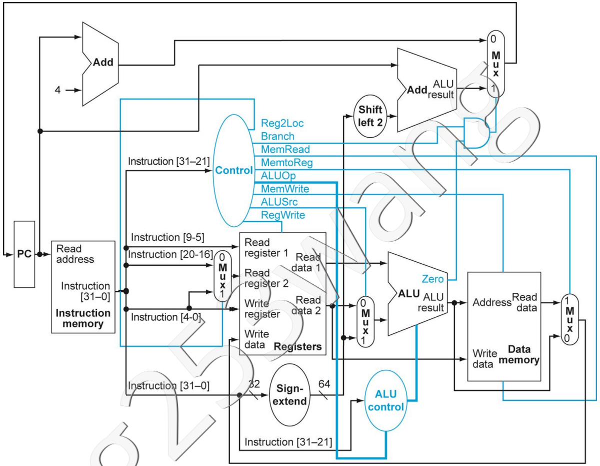


Figure 4.17 from the text

LDUR X1, [X1,#200]

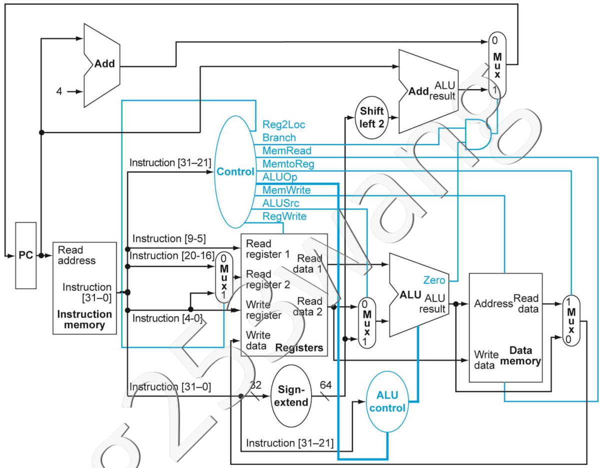


Figure 4.17 from the text

CBZ X1, #100

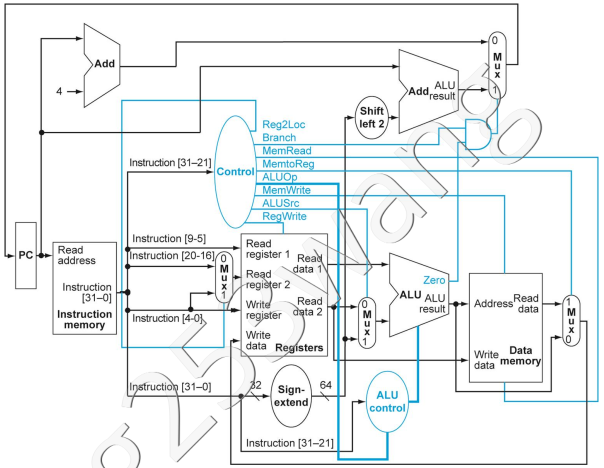


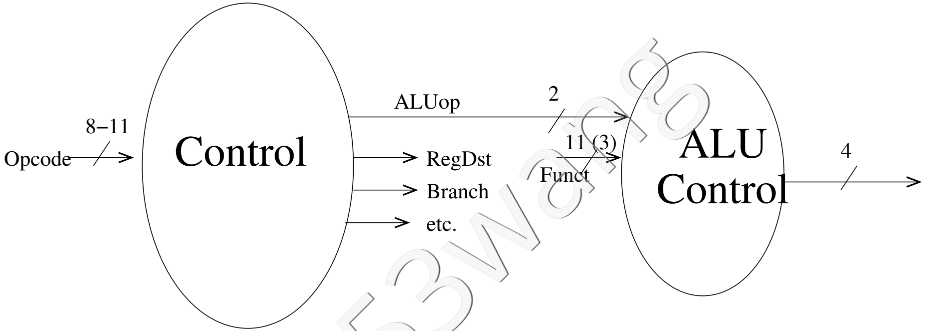
Figure 4.17 from the text

Meaning of Signals in Single-Cycle Datapath

Signal	Signal=0	Signal=1
Reg2Loc	20–16	4–0
Branch*	no branch	Branch
MemRead	no effect	memory read
MemToReg	reg write from ALU	reg write from memory
MemWrite	no effect	memory written
ALUSrc	ALU B input from reg	immediate from instruction
RegWrite	no effect	register written

* Branch is ANDed with Zero from ALU to get PCsrc
(Full version in Figure 4.16 of text.)

Overview of Single-Cycle Control



- Could be done in one level
- Multiple levels of control are conceptually simpler
- Smaller control units may also be faster
- Readings: Appendix C, section C.2*

* – may not be for ARM datapath!

Designing Single-Cycle Control

Mapping of operation to ALU control input:

Operation	ALUop	Opcode	ALU action	ALU ctrl input
LDUR	00	xxxxxx xxxxx	add	0010
STUR	00	xxxxxx xxxxx	add	0010
CBZ	01	xxxxxx xxxxx	pass b	0011
ADD	10	100010 11000	add	0010
SUB	10	110010 11000	subtract	0110
AND	10	100010 10000	AND	0000
ORR	10	101010 10000	OR	0001

Bottom four use Opcode bits to determine ALU action

ALUOp	Operation
00	Add
01	pass b
10	R-format
11	Subtract

Designing ALU Control

Operation	ALUop		Opcode											ALU	Operation			
	1	0	x	30	29	x	x	x	x	24	x	x	x	action	3	2	1	0
LDUR	0	0	x	x	x	x	x	x	x	x	x	x	x	add	0	0	1	0
STUR	0	0	x	x	x	x	x	x	x	x	x	x	x	add	0	0	1	0
CBZ	0	1	x	x	x	x	x	x	x	x	x	x	x	pass b	0	0	1	1
ADD	1	0	c	0	0	c	c	c	c	1	c	c	c	add	0	0	1	0
SUB	1	0	c	1	0	c	c	c	c	1	c	c	c	subtract	0	1	1	0
AND	1	0	c	0	0	c	c	c	c	0	c	c	c	AND	0	0	0	0
ORR	1	0	c	0	1	c	c	c	c	0	c	c	c	OR	0	0	0	1

- Remove non-varying Opcode bit positions (c in table) for ALUop=1
- Split ALU control input as Operation3,Operation2, Operation1, Operation0
  (Operation3=0 for our subset of ARM)

Implementing Main Control Function

Type	Reg2 Loc	ALU Src	Mem ToReg	Reg Write	Mem Read	Mem Write	Brch	ALU op1	ALU op0
R-format	0	0	0	1	0	0	0	1	0
LDUR	X	1	1	1	1	0	0	0	0
STUR	1	1	X	0	0	1	0	0	0
CBZ	1	0	X	0	0	0	1	0	1

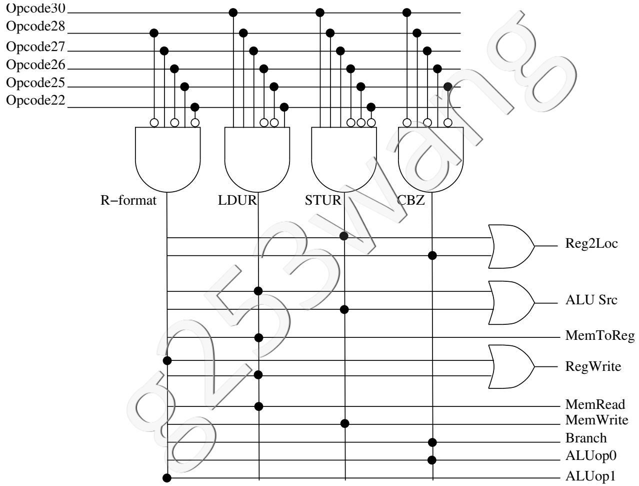
Note: MemRead is **never** don't care (multiple of 4 or crash!)

Type	Binary Opcode	Simplified Opcode
R-format	1xx 0101 x000	xxx 0101 xx0x
LDUR	111 1100 0010	x1x 1100 xx1x
STUR	111 1100 0000	x1x 1100 xx0x
CBZ	101 1010 0xxx	x0x 1010 xxxx

Simplified opcode bits: 30,28,27,26,25,22

Software generates simplified circuit, or...

Two Level Main Control Circuit



Performance of Single Cycle Machines

- Suppose memory units take 200 ps (picoseconds), ALUs 200 ps, register files 100 ps, no delay on other units
- Branch take 400 ps, CBZ take 500 ps, R-format instructions 600 ps, STUR 700 ps, LDUR 800 ps.
- Clock period must be increased to 800 ps or more
- Even worse when floating-point instructions are implemented
- Idea: use multicycle implementation and R format

Modifying the datapath

- Normally design complete datapath for all instructions together.
- Various ways to modify datapath. The following is one approach for adding a new assembly instruction:
  1. Determine what datapath is needed for new command
  2. Check if any components in current datapath can be used
  3. Wire in components of new datapath into existing datapath
     Probably requires MUXes
  4. Add new control signals to Control units
  5. Adjust old control signals to account for new command

Adding the branch command

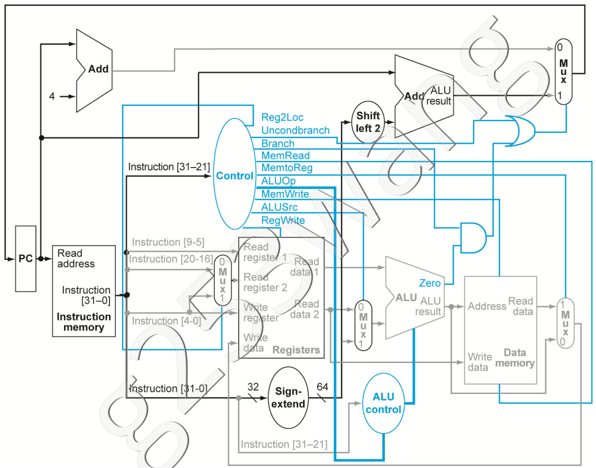


Figure 4.23 from the text

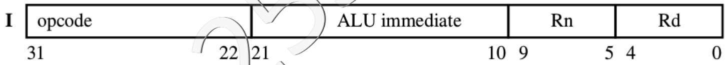
Add Branch Relative (BREL) Instruction

We will add another command, "Branch relative" (BREL) to the updated datapath.

This instruction is I-format, it branches based on a register:

`BREL Rn`

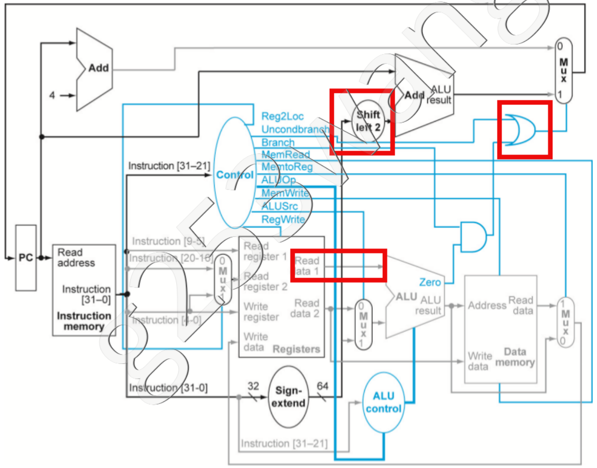
Sets PC to `PC+4*Rn`. `Rn` is in bits 9-5.



** Try this yourself. The process is explained in the following slides.

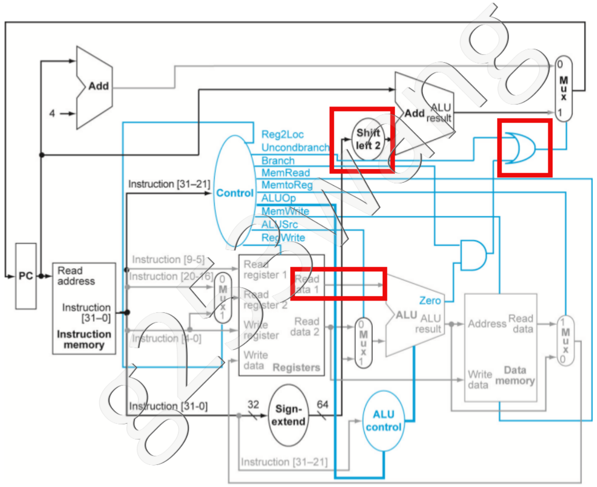
Add Branch Relative (BREL) Instruction

Add a control bit BREL. For the “Shift left 2” unit’s input, add a mux to choose between Read data 1 and Sign-extend. BREL is the select signal for this mux. If BREL=1, register Rn is given to “Shift left 2”.



Add Branch Relative (BREL) Instruction

The BREL signal also goes into the OR gated added previously.



Control Signals For BREL Instruction

- First, BREL=1, Branch=Uncondbranch=0, because BREL is used to signal we are branching instead.
- RegWrite=0, MemRead=MemWrite=0.
- Don't cares:
    - Reg2Loc=X, because we are not using "Read Register 2".
    - MemtoReg=X, because are not writing back to the register file.
    - ALUOp=XX, ALUSrc=X, because we are not using the ALU.

Remember to indicate that BREL=0 for all other instructions.

Textbook Readings

- Readings:
  - Section 4.1 – 4.4