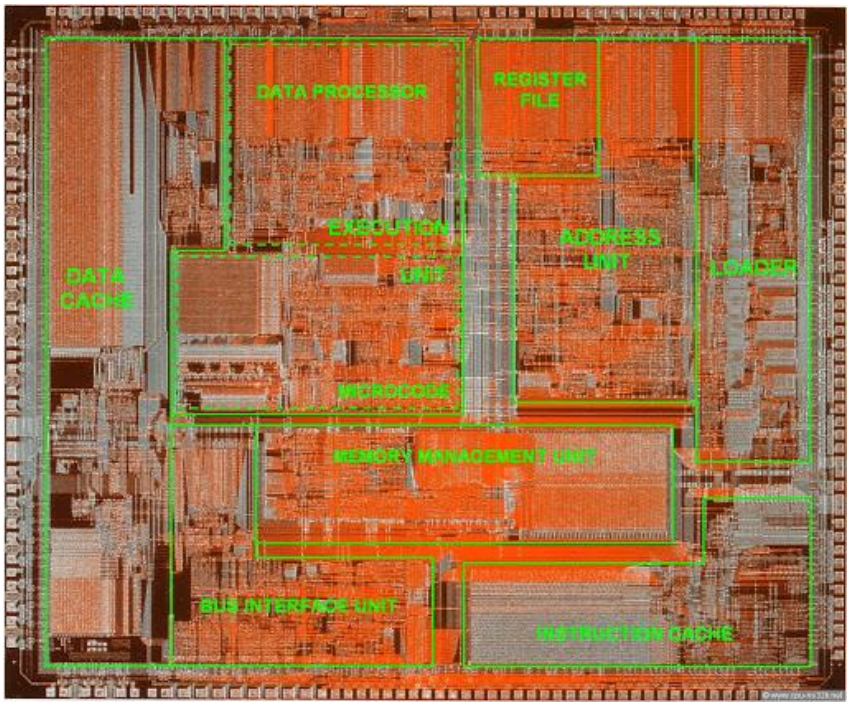
Computer Architecture
PROJECT REPORT

Made using CPUSim



Rohan Daniel - 2018A7PS0584H
Abhinav Sukumar Rao - 2018A7PS0172H
Pranav Rajagopalan - 2018A7PS0177H
Bhavyesh Desai - 2018A7PS0164H
Akul Singhal - 2018A7PS0193H

INTRODUCTION

Using Dale Skrien's CPUSim software running on JavaFX™, we attempt to build 2 Machines:

Machine - A (Part A) features a single register that is accessible by the programmer, namely an accumulator and 16 instructions.

Machine - B (Part B) features a register array of which each register can be accessed by the user. Additionally, 2 registers, namely D1 and D2 have been introduced.

ASSUMPTION

- An assumption for the m2m instruction is that both of its operands are indirect memory locations, one of them being treated as a register, namely memory address 0x02.
- Another particularly interesting assumption is that we have a multiplexer from the IR to the register array, since there needs to be a way to select a register array based on the index provided in the instruction.
- Another assumption is that we don't require the values of certain registers to be preserved during instruction executions, i.e. a few registers are "clobbered" over the course of the instruction: In part A, the accumulator, and in Part B, accumulator, D1 and D2. As a result, it is a requirement
- The reason for the above is because of the lack of either a temporary register to store the intermediate values, or the lack of a datapath between the registers.

Contributions

1. Rohan Daniel: Machine instructions for Machine A, Microinstructions for Machine B.
2. Abhinav Sukumar Rao: Microinstructions for Machine A, Machine instructions for Machine A and B, Code for Part-A-D
3. Pranav Rajagopalan: Microinstruction fine-tuning for the Machine B, part B modifications
4. Bhavyesh Desai: Part B-D Code, Report, Documentation, Stress Test
5. Akul Singhal: Part B-D Code, Report, Documentation

Opcodes and description

Instruction	Format	Description	Machine A	Machine B
hlt	hlt	Halt machine execution.	✓	✓
ipa	ipa	Read input to accumulator	✓	✓
opa	opa	Output accumulator contents	✓	✓
jmp	jmp address	Unconditional jump	✓	✓
bz	bz address	Branch if contents of accumulator is zero	✓	✓
bnz	bnz address	Branch if contents of accumulator is not zero	✓	✓
bpos	bpos address	Branch if contents of accumulator is positive	✓	✓
bneg	bneg address	Branch if contents of accumulator is negative	✓	✓
add	add address	Add value at address to accumulator	✓	✓
sub	sub address	Subtract value at address to acumulator	✓	✓
li	li value	Load immediate to accumulator	✓	✓

lda	lda address	Load value at given address to accumulator	✓	✓
sta	sta address	Store value at given address to accumulator	✓	✓
m2a	m2a address	Move value at address contained at the given addr to accumulator.	✓	✓
m2m	m2m address	Move value at address contained in 0x2 to address contained at given address.	✓	✓
labuf	labuf D1 / D2	Load accumulator to specified buffer	✗	✓
lbufa	lbufa D1 / D2	Load specified buffer to accumulator	✗	✓
lara	lara registerIndex	Load accumulator to given index of register array	✗	✓
lraa	lraa registerIndex	Load given index of register array to accumulator	✗	✓
addra	addra registerIndex1 registerIndex2	Add the contents of the two specified indices of the register array in the accumulator and store it back into position registerIndex1 of the register array.	✗	✓
subra	subra registerIndex1 registerIndex2	Subtract the contents of the second specified index the register array from the first in the accumulator and store it back into position registerIndex1 of the register array.	✗	✓
ina	ina	Increment Accumulator	✗	✓
dea	dea	Decrement Accumulator	✗	✓

SCREENSHOTS

PART A - Machine A

1. Hardware:

Type of Module: Register

name	width	initial value	read-only
ACC	16	0	☐
IR	16	0	☐
MAR	8	0	☐
MDR	16	0	☐
PC	8	128	☐
STATUS	8	0	☐

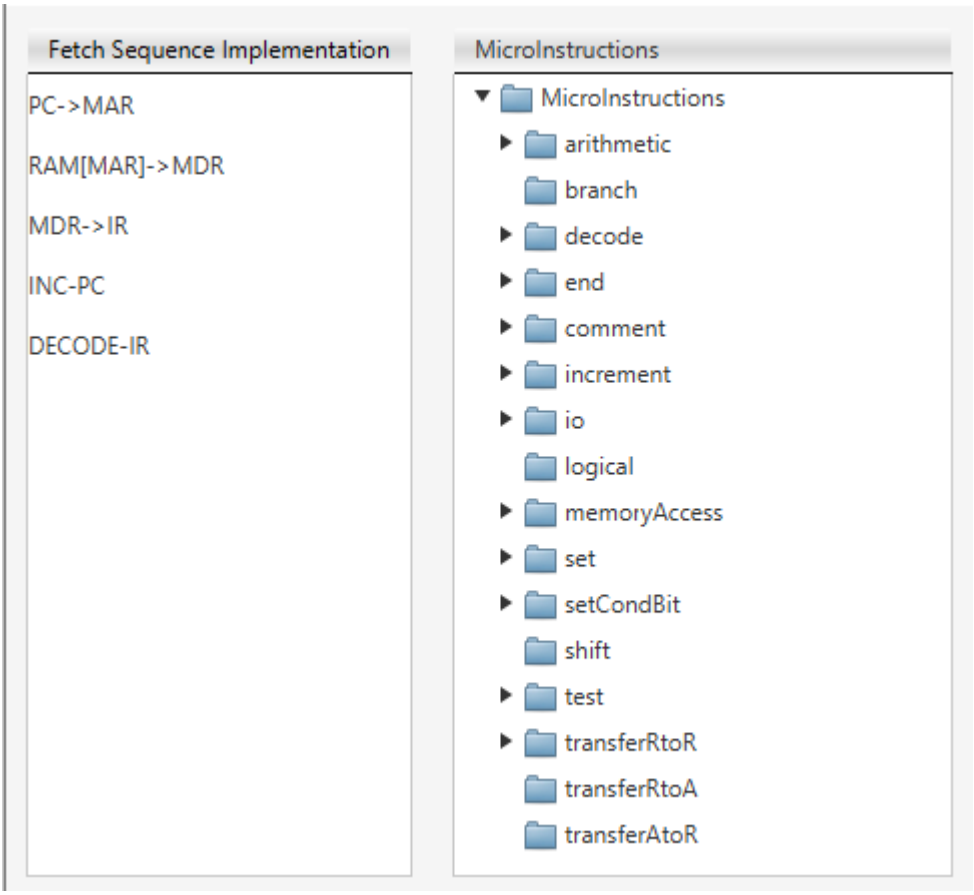
Type of Module: ConditionBit

name	register	bit	halt
carry-bit	STATUS	3	☐
halt-bit	STATUS	0	☑
overflow-bit	STATUS	2	☐
trap-bit	STATUS	1	☑

Type of Module: RAM

name	length	cellSize
RAM	256	16

2. Fetch Cycle:



3. Important Microinstructions:

Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
ACC(0-7)->MDR(0...	ACC	0	MDR	0	8
ACC(8-15)->MDR(...	ACC	8	MDR	8	8
ACC->MDR	ACC	0	MDR	0	16
IR(0-7)->MAR(0-7)	IR	0	MAR	0	8
IR(8-15)->MAR	IR	8	MAR	0	8
IR(8-15)->PC	IR	8	PC	0	8
MDR(0-7)->ACC(0...	MDR	0	ACC	0	8
MDR(0-7)->MAR	MDR	0	MAR	0	8
MDR(8-15)->ACC(...	MDR	8	ACC	8	8
MDR(8-15)->MAR	MDR	8	MAR	0	8
MDR->ACC	MDR	0	ACC	0	16
MDR->IR	MDR	0	IR	0	16
PC->MAR	PC	0	MAR	0	8

Type of Microinstruction: Arithmetic

name	type	source1	source2	destination	overflowBit	carryBit
acc*mdr->acc	MULTIPLY	ACC	MDR	ACC	halt-bit	(none)
acc+mdr->acc	ADD	ACC	MDR	ACC	halt-bit	(none)
acc-mdr->acc	SUBTRACT	ACC	MDR	ACC	halt-bit	(none)
acc/mdr->acc	DIVIDE	ACC	MDR	ACC	halt-bit	(none)

Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
ACC!=0	ACC	0	16	NE	0	1
ACC<0	ACC	0	16	LT	0	2
ACC==0	ACC	0	16	EQ	0	1
ACC>0	ACC	0	16	GT	0	2

Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
MDR->RAM[MAR]	write	RAM	MDR	MAR
RAM[MAR]->MDR	read	RAM	MDR	MAR

PART B - Machine B

1. Hardware:

Type of Module: RAM

name	length	cellSize
RAM	256	16

Type of Module: Register

name	width	initial value	read-only
ACC	16	0	☐
D1	16	0	☐
D2	16	0	☐
IR	16	0	☐
MAR	8	0	☐
MDR	16	0	☐
PC	8	128	☐
STATUS	8	0	☐

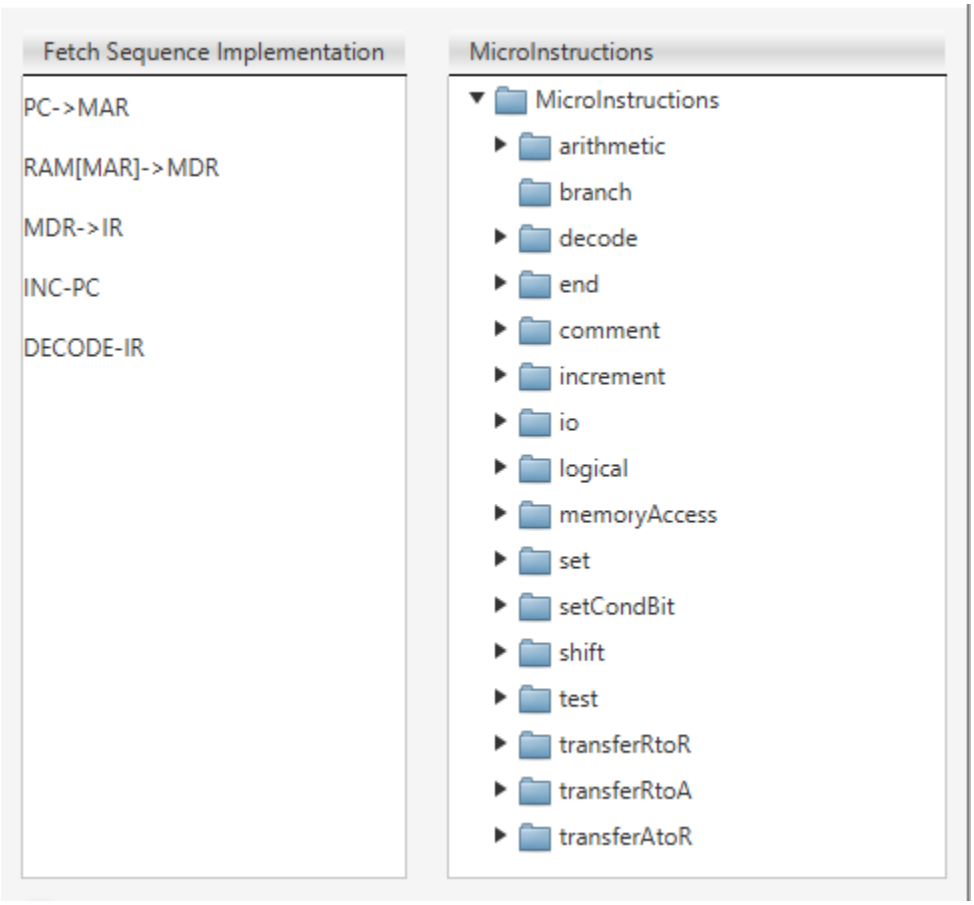
Type of Module: RegisterArray

name	length	width
RA	8	16

Type of Module: ConditionBit

name	register	bit	halt
carry-bit	STATUS	3	☐
halt-bit	STATUS	0	☑
overflow-bit	STATUS	2	☐
trap-bit	STATUS	1	☑

2. Fetch Sequence:



3. Important Microinstructions:

Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
ACC(8-15)->MDR(...	ACC	8	MDR	8	8
ACC(8-15)->PC	ACC	8	PC	0	8
ACC->D1	ACC	0	D1	0	16
ACC->D2	ACC	0	D2	0	16
ACC->MDR	ACC	0	MDR	0	16
D1->ACC	D1	0	ACC	0	16
D1->MAR	D1	8	MAR	0	8
D2->ACC	D2	0	ACC	0	16
IR(0-7)->MAR	IR	0	MAR	0	8
IR(8-15)->MAR	IR	8	MAR	0	8
IR(8-15)->PC	IR	8	PC	0	8
MDR(8-15)->ACC(...	MDR	8	ACC	8	8
MDR->ACC	MDR	0	ACC	0	16
MDR->IR	MDR	0	IR	0	16
PC->MAR	PC	0	MAR	0	8

Type of Microinstruction: TransferRtoA

name	source	srcStartBit	dest	destStartBit	numBits	index	indexStart	indexNum...
D2->RA[IR(...	D2	0	RA	0	16	IR	5	3
D2->RA[IR(...	D2	0	RA	0	16	IR	8	3

Type of Microinstruction: TransferAtoR

name	source	srcstartBit	dest	destStartBit	numBits	index	indexStart	indexNum...
RA[IR(11-1...	RA	0	D1	0	16	IR	11	3
RA[IR(5-7)]...	RA	0	D2	0	16	IR	5	3
RA[IR(8-10)...	RA	0	D1	0	16	IR	8	3
RA[IR(8-10)...	RA	0	D2	0	16	IR	8	3

Type of Microinstruction: Arithmetic

name	type	source1	source2	destination	overflowBit	carryBit
acc*mdr->acc	MULTIPLY	ACC	MDR	ACC	halt-bit	(none)
acc+d1->acc	ADD	D1	ACC	ACC	halt-bit	(none)
acc+d2->acc	ADD	D2	ACC	ACC	halt-bit	(none)
acc+mdr->acc	ADD	ACC	MDR	ACC	halt-bit	(none)
acc-d1->acc	SUBTRACT	D2	ACC	ACC	halt-bit	(none)
acc-mdr->acc	SUBTRACT	ACC	MDR	ACC	halt-bit	(none)
acc/mdr->acc	DIVIDE	ACC	MDR	ACC	halt-bit	(none)
d1*d2->d2	MULTIPLY	D2	D1	D2	halt-bit	(none)
d1+d2->d2	ADD	D2	D1	D2	halt-bit	(none)
d2-d1->d2	SUBTRACT	D2	D1	D2	halt-bit	(none)
d2/d1->d2	DIVIDE	D2	D1	D2	halt-bit	(none)

Type of Microinstruction: Test

name	register	start	numBits	comparison	value	omission
ACC!=0	ACC	0	16	NE	0	1
ACC<0	ACC	0	16	LT	0	2
ACC==0	ACC	0	16	EQ	0	1
ACC>0	ACC	0	16	GT	0	1
if(IR[7]==0)skip2	IR	7	1	EQ	0	2

CONCLUSIONS AND RESULTS

1. We managed to finish the assignment with working code and understood the need for multiple registers and datapaths, without which it may cause difficulties with implementations for instructions.
2. We understood how to work with register arrays and their benefits over individual registers.