

Teaching CPU Architecture: A New Way to Provide Effective Scaffolding

Elizabeth Patitsas, Vanessa Kroeker, Rachel Jordan, and Kimberly Voll
University of Toronto University of British Columbia Centre for Digital Media

The Paper CPU: an activity for introducing CPU architecture – and to scaffold computer simulations!

Memory and Registers

MEMORY & REGISTERS (Steps 1, 3, and 7)

As you go through the program, refer to the reference sheet at the end of this document to see where to go next when completing each step.

MEMORY

Below is a program stored in memory, beginning at memory address 0. At each memory address, one byte (2 digits) of data is stored. Memory addresses are the numbers on the top row.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

REGISTERS

Our computer has 8 registers – 8 memory slots in the cpu – to hold onto the numbers we are currently working with. Like main memory, each register has an address, shown in the left row. The columns to the right of these addresses represent clock cycles that the computer goes through.

Address	Unless a new value is written to a register, the register keeps its previous value; copy these values from the previous clock cycle.
0	0
1	0
2	1
3	0
4	0
5	0
6	0
7	0

Note: We have no register F. If you try to access register F, return a 0; if you're trying to write to register F, just move on. F is used as a flag to indicate we don't need to read or write to memory in a given instruction.

Fetch and Decode

FETCH AND DECODE (Steps 1, 2, and 6)

Fetch and Decode will ask for the instruction stored in memory, which will be 6 bytes, starting at the address PC. The value of PC will come from Execute. Parse the six bytes as follows:

- the 1st hex digit is the instruction code (iCd)
- the 2nd hex digit is the instruction function (iFn)
- the 3rd hex digit is register A (rA)
- the 4th hex digit is register B (rB)
- the remaining hex digits (8 digits, or 4 bytes) are called valC.

And then calculate the following four values: (NOTE: ALL VALUES ARE IN HEXADECIMAL.)

If iCd is ... then:	valP	srcA	srcB	dstE
1	PC + 2	F (the hex value)	F (the hex value)	F (the hex value)
3	PC + 6	F (the hex value)	F (the hex value)	rB
6	PC + 2	rA	rB	rB
7	PC + 6	F (the hex value)	F (the hex value)	F (the hex value)

PC	0	6	C
Instruction	00000000	00000000	
iCd	3	3	
iFn	0	0	
rA	8	8	
rB	2	1	
valC	1	5	
valP	6	C	
srcA	F	F	
srcB	F	F	
dstE	2	1	

Execute

EXECUTE (Steps 2, 3, 4, 5, 6 and 7)

Execute works with a lot of the rest of the computer to make sure every instruction is run properly.

- Start off by getting the following things: iCd, iFn, valC, valP, srcA, srcB, and dstE from Fetch/Decode.
- srcA is the register address where the value you will use for valA is stored. Get this value from the register, and do the same with srcB to get valB.
- Send iCd, iFn, valA, valB, valC, srcA, and srcB, over to ALU/DECIDE and wait for two values back: valE and bch.
- The nextPC will be valP, with one exception: when iCd = 7 AND bch = 1. In this case, we will move to somewhere else in our program, and nextPC will be valC. Pass nextPC to the PC cell in the next empty column of Fetch/Decode.
- Lastly, tell Memory to write the value valE to the register with address dstE.

iCd	3	3
iFn	0	0
valC	1	5
valP	6	C
srcA	F	F
srcB	F	F
dstE	2	1

Get these values from Fetch/Decode (Step #1)

valA	0	0
valB	0	0

Get these values from registers (Step #2)

valE	1	5
bch	0	0

Get these values from the ALU (Step #3)

Calculate this value (Step #4)

nextPC	6	C
--------	---	---

Finally, save valE back to the register at dstE (Step #5)

ALU and Decide Branch

ALU AND DECIDE BRANCH (Steps 4 and 5)

The job of the Arithmetic Logic Unit is to do the arithmetic and logic operations in the computer.

STEP 1 Start by getting valA and valB:

If iCd is:	srcA	srcB
1	0	0
3	valC	0
6	valA	valB
7	valC	0

STEP 2 Calculate a new hexadecimal value, valE:

If iCd is:	and if iFn is ... then:	valE
-6	0, 1, 2	valB + valA
6	0	valB - valA
6	1	valB * valA
6	2	valB / valA

STEP 3 Your job is also to figure out if the conditions are right for branching. Execute will then decide if we are actually going to branch.

If iCd is:	and if iFn is ... then:	bch
-7	0	0
7	0	jump unconditionally: bch = 1
7	1	If valE < 0 in the previous clock cycle, then bch = 1. Otherwise, bch = 0.
7	2	0

Get these values from Execute:

iCd	3	3
iFn	0	0
valA	0	0
valB	0	0
valC	1	5

Then calculate these:

valA	1	5
valB	0	0
valE	1	5
bch	0	0

The Paper CPU Activity

How it works:

- Students do the activity in groups
- Each student acts as one of the four stages
- Students calculate and pass values around

Goals:

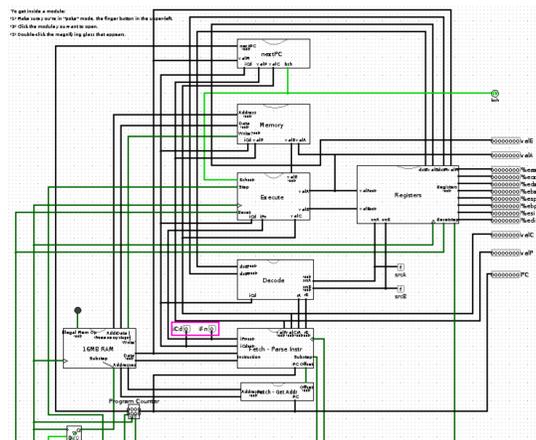
- Introduce CPU architecture
- Determine what a mystery program does

How we implement it:

- The activity is used as an introduction to CPU architecture
- It provides scaffolding for teaching a simulated Y86 CPU in Logisim
- At UBC we run the Paper CPU in lab followed directly by the Logisim simulation
- At UToronto we do the activity on its own in tutorial

Our Logisim Simulation

- Previously, we taught CPU architecture by starting with a circuit simulation of a "simple" CPU
- Students found even the simplest version of it overwhelming!
- We created the Paper CPU as an activity to scaffold the Y86 simulation



Advantages of the Paper CPU

Effective scaffolding: students become comfortable with the structure of a CPU

Learning for the whole class: jumping to Logisim only benefited the top students

Discovery learning: the students typically "discover" pipelining and data forwarding on their own.

Collaborative learning: by putting students into a group to collaborate.

Little overhead: little to no extra teaching load; reduces student questions later on.

Notes for afterwards: students have a paper copy of their work to add to their notes for the class.

Improved student buy-in: we survey our students after every lab; since adding the activity student feedback has improved significantly!

You can do it too!

The activity is freely available at <http://www.ugrad.cs.ubc.ca/~cs121/2011W2/Labs/Lab9/playcpu.pdf>

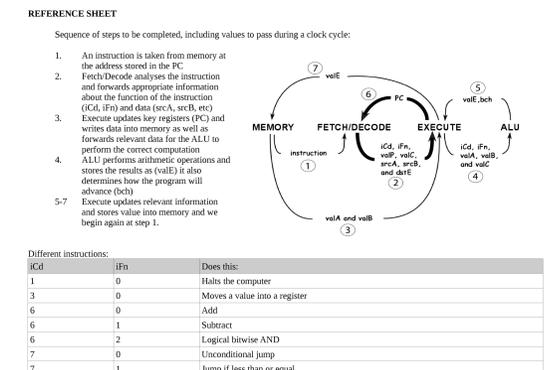


Figure: Reference sheet for the Paper CPU

Acknowledgements

- The Paper CPU has been developed with the feedback of UBC's CPSC 121 students and TAs – a big thank you to them!
- The Y86 Logisim simulation was made by Patrice Belleville and Steve Wolfman, based on Bryant and O'Hallaron's Y86 architecture.
- Development was partially funded by UBC's CS Science Education Initiative.
- E.P. is supervised by Steve Easterbrook and Michelle Craig; travel funding from NSERC.