Mark 2 FORTH Computer

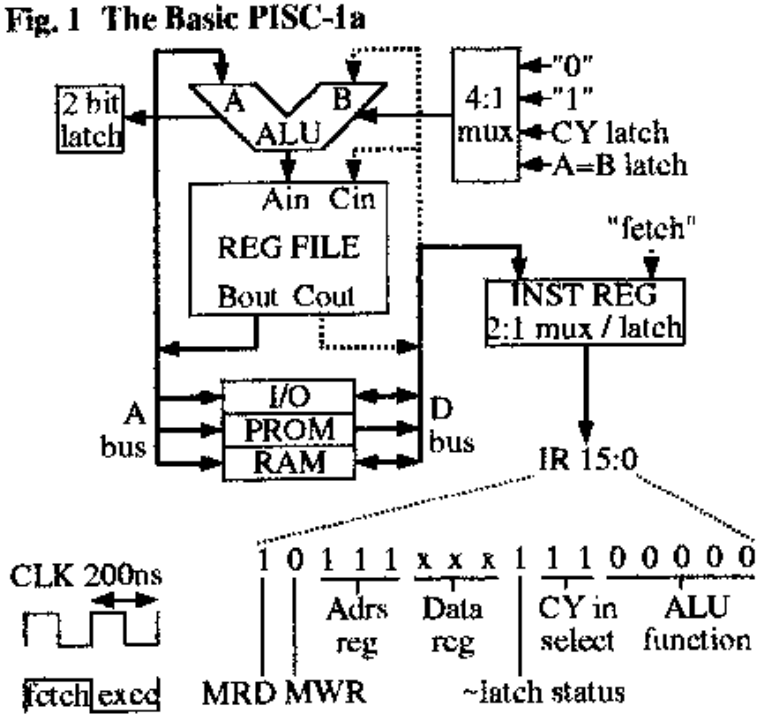
Architecture Photos Schematics Back to projects

Specification

Technology	Bitslice / PLD
Registers	16
Data width	16-Bit
Address width	16-Bit
Software	Fig-FORTH

Inspiration

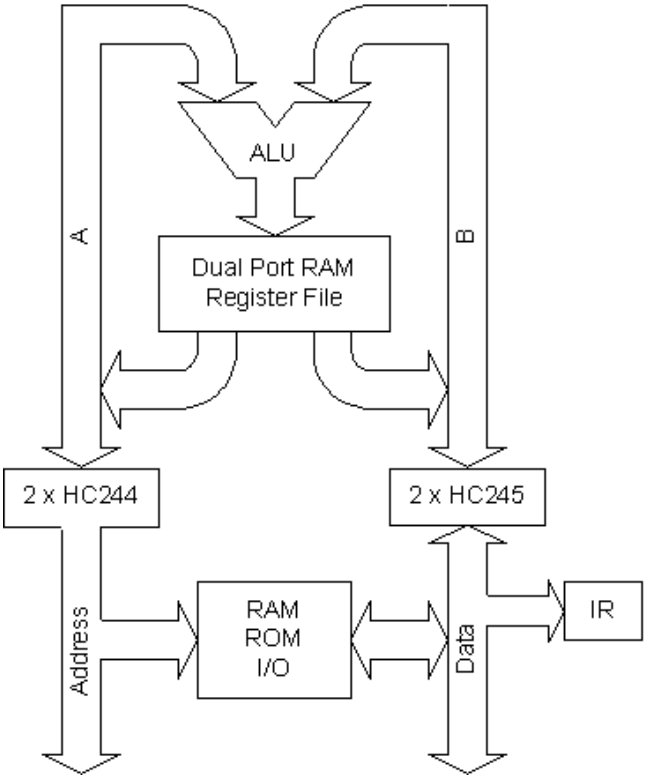
The Mark 2 was inspired by the PISC (Pathetic Instruction Set Computer) described by Bradford J. Rodriguez of McMaster University in his article A Minimal TTL Processor for Architecture Exploration.



I came across the PISC whilst building my Mark 1. The PISC appeared to be not only simpler but more powerful and I felt a bit foolish. When I tabulated their comparative performance cycle-for-cycle, the Mark 1 wasn't so bad after all.

CPU Architecture

Whilst the original PISC 1 used the rare and obsolete 74172 register file, the Mark 2, like the PISC 2, uses the also obsolete but not quite so rare Am29705 4-bit by 16 word Dual Port RAM - part of the AMD 29000 Bitslice family.



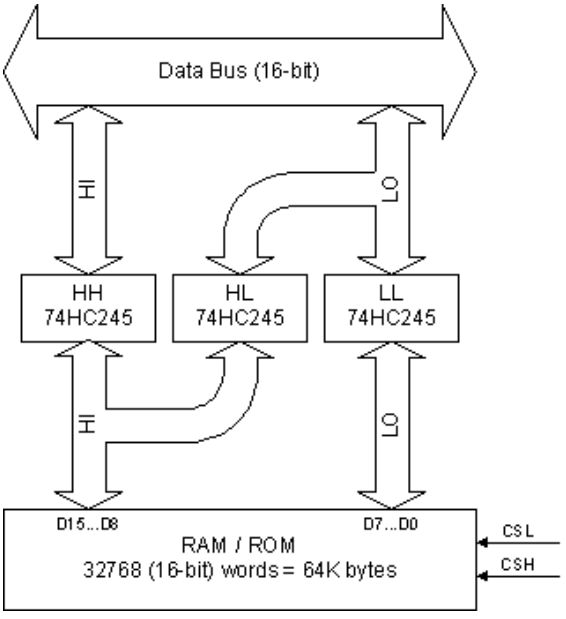
IR = Instruction Register

The ALU bottleneck was the worst feature of the Mark 1. Here, the ALU sits between the register file and the system buses.

Instruction execution takes two cycles: FETCH and EXECUTE. The program counter (PC) is placed on the address bus during FETCH and IR is loaded from memory. Simultaneously, PC is incremented (by 2). The ALU is used to increment the program counter, which can be any of the general-purpose registers.

Byte/word addressing

The Mark 2 is a 16-bit machine. Words are aligned on even address boundaries. To support byte addressing, there are three HC245 bus transceivers on the memory board. The middle (HL) transceiver is used to access bytes at odd addresses; and the lower (LL) to access bytes at even addresses. 00H is forced onto the upper half of the data bus during byte reads.



PISC memory was organised as words on unit address boundaries with no support for byte addressing. This made it easy to increment the program counter during a FETCH. The 74181 has an "A plus Carry" function. By asserting the carry input, this function is effectively "A+1". It is not so easy to perform "A+2".

A spare section of a 74HC244 is used to force the value 0010_2 onto the lowest 4-bits of the B bus during FETCH. The function "A+B+Carry" is used on the lower 181 (with no carry-in). The function "A+Carry" is used on the other three allowing the upper 12 bits of the B bus to be left floating.

Instruction set

The Mark 2 has a highly encoded instruction set. I don't consider it a microcoded machine.

Operation	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
No Operation	0	0	0	0												
IRQ Enable	0	0	0	1												
IRQ Disable	0	0	1	0												
Memory Read Word	0	0	1	1					RB				RA			
Memory Read Byte	0	1	0	0					RB				RA			
Memory Write Word	0	1	0	1					RB				RA			
Memory Write Byte	0	1	1	0					RB				RA			
Register - Register	0	1	1	1	ALU Function				RB				RA			
Test Register	1	0	0	0					RB							
Test Memory Word	1	0	0	1									RA			
Immediate	1	0	1	0	ALU Function				RB				D3	D2	D1	D0
Conditional branch	1	0	1	1		T/F	Flag				*D5	D4	D3	D2	D1	D0
I/O Read Byte	1	1	0	0					RB				RA			
I/O Write Byte	1	1	0	1					RB				RA			
Branch if IRQ	1	1	1	0							*D5	D4	D3	D2	D1	D0
Bus Reset	1	1	1	1												

* Sign-extended

Assembler

Mark 2 assembly language looks weird. It's implemented using MASM macros:

```
RdWord    MACRO rb, ra
RdByte    MACRO rb, ra
WrWord    MACRO rb, ra
WrByte    MACRO rb, ra
R2R       MACRO alu, rb, ra:=<0>
TestR     MACRO r
TestM     MACRO r
Immed     MACRO alu, rb, d
Cond      MACRO cond, dest
IOR       MACRO rb, ra
IOW       MACRO rb, ra
```

I also make extensive use of the following:

```
$PUSH     MACRO rb, sp
          immed alu_sub, sp, 2
```

```
          wrword rb, sp
          ENDM

$PULL     MACRO rb, sp
          rdword rb, sp
          immed alu_add, sp, 2
          ENDM
```

`alu_add` and `alu_sub` are ALU function codes.

The source code is available for download here:

Mk2.INC	MASM include file declaring constants and macros
Monitor.ASM	ROM Resident monitor for downloading code via the serial port
Forth.ASM	fig-FORTH †
Split.cpp	Split binary into odd/even ROM images
m.bat	Batch file to automate assembly and splitting
m4000.bat	Assemble for download to RAM at 4000h

The bootstrap code which brings the system up following a hard reset can be found at the beginning of Monitor. The initial program counter is 0000h. NEXT is located at address 0004h within the range of an absolute jump (load PC immediate).

† Stack ordering of remainders and quotients left by division words is reverse of standard fig-FORTH.

Programmable logic

Programmable logic devices (PLDs) are used everywhere: three on the controller, and one on each of the other cards.

On the controller, at the heart of the system, 22V10.SEQ is the state machine which controls the FETCH-EXECUTE cycle (amongst other things). 22V10.ALU and 22V10.REG assist with instruction decoding. The former, as its name suggests, is concerned with generating control signals for the ALU. The latter controls the address select inputs of the register file. Sometimes, these control signals are derived from the instruction register (IR) and simply pass transparently through the PLD. Other times, e.g. during FETCH, the PLD supplies specific codes.

Instead of conventional control signals like MR and MW, **everything** is controlled by the imaginatively named 'BUS' bus. This 4-bit code, generated by 22V10.SEQ, and fed to every other PLD in the system, is nearly the same as the instruction set:

0000	Float	All bus signals go tri-state
0001	Start	Load PC with 0000h
0010	Fetch	FETCH cycle
0011 - 1110	Execute	Same as $IR_{15...12}$
1111	Reset	Hardware reset

The PLD source and compiled JEDEC files are available for download here:

Include file Ops.INC
22V10.SEQ PLD JEDEC Controller
22V10.ALU PLD JEDEC
22V10.REG PLD JEDEC
16V8.MEM PLD JEDEC Memory
16V8.CPU PLD JEDEC CPU
22V10.SER PLD JEDEC I/O

Backplane

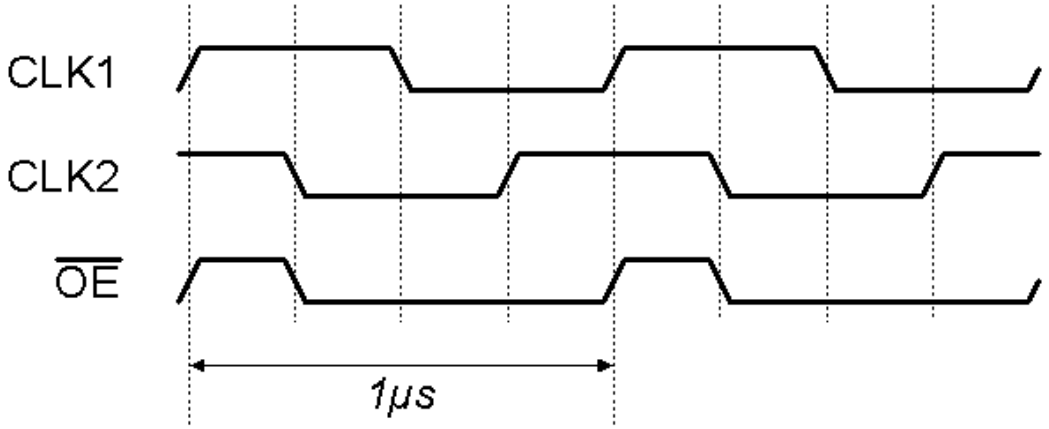
I've again used a 64-way DIN 41612 backplane.

	A	C
1	+5V	
2	CLK1	
3	S0	0V / Screen
4	S1	FLAG1
5	S2	FLAG2
6	S3	FLAG3
7	M	BUS0
8	CARRY	BUS1
9	BUS3	BUS2
10	A0	D0
11	A1	D1
12	A2	D2
13	A3	D3
14	A4	D4
15	A5	D5
16	A6	D6
17	A7	D7
18	A8	D8
19	A9	D9
20	A10	D10
21	A11	D11
22	A12	D12
23	A13	D13
24	A14	D14
25	A15	D15
26	RA0	RB0
27	RA1	RB1
28	RA2	RB2
29	RA3	RB3
30	0V / Screen	IRQ / FLAG0
31	CLK2	
32	0V	

I've successfully clocked the Mark 2 up to 4 MIPS via the external clock input (16 MHz divided by 4) but the bus signals do not look pretty at that speed. The bus is not terminated and it has no ground plane! I generally run it at a conservative 1 MHz as I do the Mark 1.

Bus contention

I had some RAM corruption problems during development due to tri-state bus contention. Basically, I was not allowing the bus to go tri-state between cycles. The outputs were fighting one another for a few tens of nanoseconds producing spikes on the bus and power supplies. Due to propagation delays, de-selected outputs take a while to go tri-state. What's worse, unequal propogation delays through the logic cause weird things to happen while BUS is changing. Much of the decoding is essentially asynchronous combinatorial logic. The solution was to hold off output enable on the memory board for the first quarter of each cycle. This allows time for everything to settle down:



Fortunately, I was able to make this change by modifying the CUPL hardware definition language (HDL) in the 16V8 GAL on the memory board.

Comparative performance

The Mark 2 is not that much faster than the Mark 1. Having twice the data width gives it a factor of 2 advantage, which it promptly throws away by taking two cycles to execute an instruction (FETCH-EXECUTE). However, it has more registers to play with and it wipes the floor with the Mark 1 at multiplication and division. The following table compares the number of cycles consumed by each FORTH primitive:

	Mk1	Mk2	Diff
Enter	14	16	-2
;S	12	12	0
LIT	14	16	-2
EXECUTE	7	6	1
(DOES)	19	22	-3
BRANCH	14	10	4
0BRANCH	21	16	5
(LOOP)	* 29/32	* 24/26	
(DO)	19	24	-5
LEAVE	15	16	-1
R>	13	16	-3
>R	13	16	-3
R	12	14	-2
AND	19	18	1
OR	19	18	1
XOR	19	18	1
+	20	18	2
-		18	
0=	19	16	3
0<	17	16	1
DUP	14	14	0
SWAP	21	20	1
DROP	10	10	0
OVER	16	16	0
@	14	14	0
!	16	18	-2
C@	13	14	-1

C!	14	18	-4
D+	37	32	5
NEGATE	17	16	1
DNEGATE	25	26	-1
U*	* 325/613	* 118/182	
U/	531	* 190/222	

* *Min/Max*

The Mark 1 is actually faster at 32-bit negation! Doh! Ah well, never mind, back to the drawing board ...

---

andrew@aholme.co.uk