25x Emulator

Chuck Moore

chipchuck@colorforth.com

2001 November

Abstract

The 25x is multi-computer chip. A 5x5 array of c18 computers on a 7sq-mm die. It connects to an 18-bit SRAM or SDRAM memory chip and has serial links to the real world. The computers communicate with each other over parallel buses.

This architecture suggests a new factoring of parallel tasks. Each computer specializes in a simple one, limited by its 192 words of memory. Computer 00 reads SRAM, computer 40 drives a crystal, etc.

A simulator for the c18 computer is programmed in colorForth. It can simulate as many of 25x's computers as desired. Fast enough to debug applications. Graphic displays of state present more information than available with a real chip.

c18 emulator

Each c18 has a block (256 words) of Pentium memory for its memory and registers. RAM is words 0-7f, ROM is 80-bf, registers are c0-db and dc-ff unused. It is presumed that the c18 cross-compiler has been run to load ROM. The word 'reset ' sets registers undefined and starts executing code from ROM.

Each emulator step is 1 instruction cycle, 400 ps. An on-chip memory access takes 3 cycles. An off-chip access to 4ns SRAM takes 10 cycles. An add instruction requires arguments stable for 2 cycles, to allow carry propagation.

The state of the computer is the contents of its registers. The stacks are displayed in green, the instruction register and address in white and the address registers in blue. 25 computers can be emulated. Of these, 0-6 can be displayed. This is how the display of 2 computers appears at power-up:

```
.....  .....
.....  .....
.....  .....
.....  .....
.....  .....
.....  .....
.....  .....
.....  .....
.....  .....  return stack

.....  .....  b
.....  .....  a

00080  00080  pc
.....  .....  instruction

.....  .....  t
.....  .....  data stack
.....  .....
.....  .....
.....  .....
.....  .....
.....  .....
.....  .....
.....  .....
```

The dots indicate undefined registers. The pc and instruction registers are initialized by reset. Thereafter, the display is updated after each emulate command. The computers start identically, but their ROM is different. Computer 0 reads Flash and loads its program. That directs it to read more Flash and distribute programs to the other computers.

25x

The 25x is an array of c18 computers. Each is connected to 2 parallel buses. Software configures some as bus masters that send or request data from others configured as slaves.

The c18 has a register connected to each bus. Data written to the register will be placed on the bus when the computer is selected to send. If the register is full, an attempt to write will be delayed until it's empty. Reading the register will be delayed until it's been selected and has received data.

The actual protocol of sending and receiving is included in the application code distributed by the master. Thus the address and length of a message is known to both master and slave and need not be communicated. Only a select word with the slave's address is needed to send or receive. This address is a single bit, so broadcast messages are possible.

Emulator

A c18 emulator is pretty simple: a loop that unpacks instructions and executes a 32-way jump to code that emulates each instruction. Measuring execution time would add some complication.

The 25x emulator is more demanding. With several independent computers, timing becomes important since it controls their synchronization. A loop will execute 1 instruction cycle for each computer. Then repeats. If an instruction takes more than 1 cycle, it must be delayed. This approximates the asynchronous behavior of the computers.

A variable **phase** is used to accomplish this. It starts at 0, counts up the required time, then executes the instruction. Most instructions don't need a delay. Those that do:

- 1 cycle delay: **+*** to propagate carry 9 places
- 2 cycle delay: **+** to propagate carry 18 places
- 3 cycle delay: **n @ @+ @b !r ! !+ !b** memory access time

In addition, fetching the next instruction word takes 3 cycles, starting after the last memory access in the current word. So each computer is executing a particular phase of its particular instructions.

The computers are synchronized by exchanging data. The emulator so far has a single bus. **@b** with address 100 will wait until bit 18 of the bus is 1, indicating data present, before reading it. It then sets b18 to 0, to acknowledge receipt. Likewise, **!b** will write to the bus when b18 is 0, and set it to 1. These cause indefinite delays and might actually hang. The actual computer has a time-out, but this is not emulated.

Emulator code

Here is a sampling of Pentium colorForth code for the Emulator. This is documentation for a work in progress, so typos and changes can be expected. See www.colorforth.com for the latest version of pre-parsed source. Colors are here represented by the conventions used in the c18 Compiler paper.

```
INST n   1 slot +! 1f and jump
   ret nul call jmp  nul nul jz jns
   n @+ @b @a  !r !+ !b !a
   +* 2* 2/ -  + and or drop
   dup over pop a  nul b! push a!

S0   ir @ 8192 / inst ;
S1   ir @ 256 / inst ;
S2   ir @ 8 / inst ;
S3   ir @ 4 * inst ;
S4   cl @ 0 or drop -if pc @ mem @ ir ! 1 pc +! set 0 slot ! s0 ;
   then ;
```

Ir and **cl** are indexed by **com**, the current computer. Thus they aren't variables, though similar. **set** sets **cl** to 3, from whence it's counted down. **S4** fetches the next instruction word.

```
TICK   -1 cl +! slot @ jump s0 s1 s2 s3 s4
T   bus+ @ bus ! nc for i com ! tick -next ;
TICKS n   for t next ;
```

Bus+ and **bus** are variables. **Nc** is the number of computers. Computers read **bus** and write **bus+** in order that all use the old value before setting a new one.

Here is the code for some instructions:

```
2*   t @ 2* 3ffff and t ! ;
+*   1 cnt  t @ 1 ? if  s @ +  then 2/ t ! ;
+    2 cnt  -t t @ + 3ffff and t ! ;
@+   3 cnt  ar @  1 ar +!  mem @ +t  set ;
```

Here **-t** pops a number off the emulated data stack, and **+t** pushes one on. **Cnt** returns if **phase** matches the stack, otherwise returns to the calling word using **pop drop**.

25x code

Here is sample code for a c18 downloading code from the bus. It is compiled into ROM and executed at power-up.

```
90 org
WORDS na   a! begin  @b !+  -1 + until drop ;
80 org  100 b! 0  63 over words  push ;
```

Words reads from the bus register and writes to memory, starting at 0. **Until** counts the stack down to -1. Finally a jump to 0. Pretty simple boot code.