History:

3/29/2021

First stab at documenting. Needs instructions all written out and proof-reading on whatever I just wrote.

Overview

CPU-16 is a relatively small 16-bit CPU. It is a load/store architecture without any pipelining to keep the size down. On an Altera Cyclone it comes in at around 500 LE's. This goes down to about 400 LE's if the shifter isn't multi-bit and performs one bit-shift per instructuin

I've added a flag that allows you to switch out the on-chip memory but that will add an eyewatering 400 LE's or thereabouts. So if you have block-ram or something similar, use that.

Most instruction take 3 machine cycles. A machine cycle is 2 clock cycles, so each instruction takes 6 clocks cycles. The main reason for this is that block-ram registers the address and data, which would increase latency. If you're using registers, that penalty goes away. Come to think of it, adding an additional state to insert a bubble to write registers back would bring it to 4, regardless of whether it's block ram or registers.

Other factors include access to memory. A simple external memory module would decrease the speed more but again, if you're using block-ram, you'll be fine.

Both the address- and databus are 16-bit wide. However, accessing bytes will mean that the physical address space goes down to 32Kx16 bits.

Interrupts and bus arbitration mechanism are provided.

Instruction Encoding

Instructions are grouped into the familiar groups:

- Register
- Immediate
- Load / Store
- Branch

Register

OPCODE	DST	SRC	SIZE	IMMEDIATE
00000	R0-7	R0-7	-	-----

Immediate

OPCODE	DST	8-BIT UNSIGNED IMMEDIATE
00000	R0-7	NNNNNNNN

Load / Store

OPCODE	DST	SRC	SIZE	UNSIGNED OFFSET
00000	R0-7	R0-7	B/W	NNNN

Branch

OPCODE	DST	10-BIT SIGNED PC-OFFSET
00000	R0-7	NNNNNNNNNNN

Conditional Branch

OPCODE	CONDITION	7-BIT SIGNED PC-OFFSET
00000	0-7	SNNNNNNN

Register and Immediate instructions are straight-forward and don't really need further explanation aside from that the immediate values are unsigned. Logical instructions such as AND, CMP and others will have their upper 8-bits cleared.
To enable full-16 bit length a 'load-upper-immediate' instruction is provided that does the opposite and loads the immediate 8-bit value in the upper 8 bits. To construct a 16-bit value therefore requires two instructions: One LUI and one ADD or OR instruction to fill the entire word.

Load and store both can ONLY reach forward locations from the source register up to 16-bytes or 8 words. The byte/word flag determines the size of the data read / written. The memory interface provides two strobes (lb_n and up_n) to interface to standard SRAMs.

Branches are a bit of a mixed bag. The most straightforward JR (Jump Register), which moves the value from the register directly to the PC and JALR (Jump-and-Link Register) which moves does the same as above stores the address of the instruction after the jump in R6. This register is hardwired and care therefore needs to be taken to preserve its contents.

JR and JALR are both unconditional and can span the entire address space. By using JALR, subroutines can easily be created by using the value in R6 to return to the address after the original jump. Nesting subroutines can be created by creating a stack frame and storing / retrieving the return address.

While any register can be used as a stack frame, it's useful to dedicate a single register to this, which is usually the last register in the register file; In this case R7.

Two other variants for JR and JALR are J and JAL respectively. These functions take the 10-bit signed word offset from the instruction and add the to current PC. This allows for short distance calls or branches.

Conditional branches hijack / overlay the immediate instruction format where instead of the destination register, this field holds one of eight conditions to act on. If true, the normally unsigned 8-bit immediate value will be converted to a signed one and allows branches to be in the range from -128 to 127 words relative to the PC.

Most of the time, this suffices as 128 words is quite a few instructions. For larger distances, jump-tables or loading register with immediate values can alleviate this.

Conditional Flags

The CPU keeps track of 4 conditional flags: Zero, Negative, Carry and Overflow. Each operation potentially affects these flags. For more information on what each instruction modifies, see the instruction opcodes below. An addition interrupt enable flag can be set using the MOVSR instruction.

The MOVSR instruction allows bits to be set / cleared using a single instruction. When bit 7 of the immediate value is set, bits that are set in the lower 5 bits will be SET in the SR. On the flipside (pun intended), clearing bit 7 will clear each bit set in the lower 5 bits in the SR.

Thus, the value of bit 7 determines whether or not the bits specified will be set or cleared. Note that multiple flags can be changed with a single operation.

Exceptions

The CPU allows up to 8 exceptions. An external 'true' interrupt is brought out through the IRQ pin. Active positive, both the PC and SR are backed up into two temporary registers, creatively called EPC and ESR. On encountering the RFE (Return-From-Exception) instruction, these flags are copied back to their respective destinations and execution continues.

Note that the IRQ instruction can take up to 3+1 machine cycles. An intack pin would make sense to add at some point now that I'm writing this. Alternatively, ensuring that the irq pin stays asserted for a number of cycles works as well.

Interrupts can be enabled by setting / clearing the Interrupt (I) bit in the SR.

Outside the external interrupt, the SWI (Software Interrupt) can be used to jump to up to 8-fixed locations. Note that when using software interrupts both the PC and SR are NOT saved. Executing an RFE would therefore mess things up nicely.

SWI's are therefore most useful to overcome some of the shortcomings of the size of the immediate values. Passing in a value before calling an SWI and using that as an index into a jump-table allows an easy way to overcome this limitation.

Bus Arbitration

A simple bus mastering scheme is implemented where the CPU can effectively be halted and release the bus and control signals. Two external signals, BUSREQ and BUSACK can be used to halt the CPU and have it 'spin' until BUSREQ has been released.

This makes it simple to share memory with other subsystems. The time to acquire the bus will be the same as that of the external interrupt to make sure the CPU is in the right state before handing over control.

Note: Interrupts have a lower priority than bus requests. Therefore, if the interrupt bit is set together with a BUSREQ, the interrupt will be delayed until the bus relinquishes control.

Opcodes

SUB Rd,Rs

OPCODE	DST	SRC	SIZE	IMMEDIATE
00000	R0-7	R0-7	X	XXXX

Operation:	Subtract Rs from Rd, storing result in Rd
Mnemonic:	SUB Rd,Rs
Cycle Count:	3
Flags affected:	-NCZV