COL216 Computer Architecture

Lab Assignment 2 Stage 8: `BL`, `SWI` instructions and predication

At this stage the remaining parts of the processor design need to be filled in. This includes implementation of function calls, exceptions, predication and input/output. The scope of work is defined in such a manner that several important concepts are being covered, while keeping the complexity very low.

Control transfer in bl and swi instructions

Both `bl` and `swi` instructions require `pc` to get a new value while saving the old value in a link register. New value for `bl` is specified in same way as `b` instruction. For `swi`, this value is the address 0x00000008.

Saving return address

In standard ARM architecture, exceptions use link registers that are different from user's lr (`r14`). Moreover, each exception has a separate link register. This is particularly useful if an asynchronous exception comes just after a function call (`bl` instruction) before `lr` is saved in stack or somewhere else. Since we are not dealing with any asynchronous exceptions, we can work with a single link register (`r14`). Thus return address will be saved in same register in `bl` as well as `swi`. Note that swi instructions can be present in normal routines/functions and bl instructions can be present in interrupt service routines (ISRs). Both these situations have to be dealt with as nested function calls, requiring saving of `lr`. This is to be done in software. Hardware design need not be concerned with this nesting.

Need for new instructions - ret and rte

Return from a normal routine or ISR is usually done by transferring return address to `pc` (`r15`) using a DP instruction or `ldmfd` instruction. Since we have chosen to keep `pc` independent of `r15`, we need to define special instructions for this purpose. Let these instructions be `ret` (return from normal routine) and `rte` (return from exception). Formats for these are described in a later section.

Modes

There will be just two modes - user mode (a non-privileged mode) and supervisor mode (a privileged mode). Instruction `swi` will cause mode to change from user to supervisor and instruction `rte` will cause mode to change from supervisor to user. Instructions `bl` and `ret` cause no mode change.

RET is return from bl
RTE is return from exception

Address space and protection

The address space is to be partitioned into user area and system area. The system area will house the ISRs and I/O addresses. This area will be accessible in privileged mode only.

Treating reset as exception

In ARM reset signal is also considered as exception, resulting in transfer of control to address 0x00000000 and setting the mode to supervisor mode. Logic for making `pc` = 0 on reset signal is already there in the design done so far. Mode initialization needs to be added. What about interrupt service routine (ISR) for this exception? Normally a branch instruction would be placed at 0x00000000, which would take the control to the beginning of the operating system. Since in our case there is no OS, the control needs to be transfered to the user program. Assume a fixed starting address of user program, for example, starting of the user area. The ISR for reset just needs to load this address into `r14` and then use `rte` instruction to transfer control to the user program and change the mode to user mode.

CPSR and other registers

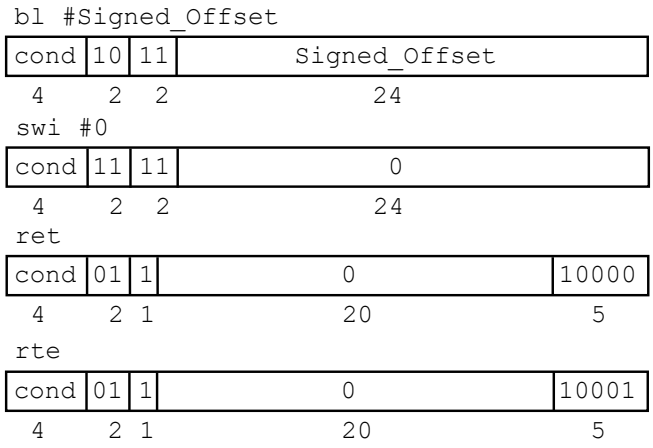
We do not need a full fledged CPSR. Apart from the four flags that are already there, only a mode flag needs to be introduced. Further, the additional registers for supervisor mode (`r13`, `r14` and SPSR) are not to be implemented. This means that the flags will not be saved on `swi` exception and a common stack will be used in both modes.

I/O function

A byte-wide input port is to be implemented. It would be the responsibility of the test-bench to supply a sequence of bytes to this port . ISR of swi will provide read function for this port. This will be non-blocking read, that is, read will be performed irrespective of the status of data availability at the port. From the port 9 bits (a data byte and a status bit) will be read using instruction `ldr` or `ldrh` and returned to user program. If the status bit indicates that data was not available, the user would discard the data byte and try again.

Instruction formats

Adjoining figure shows the formats of the 4 instructions to be implemented at this stage. Bits 23-0 of swi instruction can be ignored. Formats for `ret` and `rte` are actually undefined for the ARM version we have considered and are reserved for later versions. In these formats bits 27-25 are identical to DT instructions with register specified offset, but bit 4 distinguishes the two ('1' in case of the new instructions



defined here and '0' in case of DT with register specified offset). For these instructions, bits 24-5 may be ignored.

Full predication and S-bit

Full predication means that all instructions are affected by the condition specified by bits 31-28 of the instruction, not only branch instructions. At this stage full predication is to be implemented with complete set of conditions. Condition checking logic is already there in the current design. Output of this logic is to be used for enabling/disabling control signal for RF write, Memory write and flags update. Also s-bit for DP and multiply group of instructions is to be used to control flags update.