332 Advanced Computer Architecture

Unassessed tutorial exercise 1: Instruction set issues in pipeline design

During the gruelling interview for the post of Deputy Technical President, IC Inside^{TM} Computers Incorporated, you have been asked to investigate a new addressing mode for a MIPS-like architecture. The idea is to allow one of the operands of an ALU instruction to be in memory. To offset this increase in complexity, you restrict all memory addressing (in Load, Store and ALU instructions) to be register-indirect only (i.e. no displacement addressing).

For example, suppose `R1` is an index into an array starting at address 100, and we need to multiply the `R1`th element by 123. On MIPS you could write:

```
LD   R2, 100(R1)
MULI R3, R2, 123
```

Here we load `R2` with the element of the array, using displacement addressing. With the modified architecture, you would have to write

```
ADDI R2, R1, 100
MULI R3, (R2), 123
```

Here, the first instruction does the displacement calculation explicitly, leaving a pointer to the array element in `R2`. The second instruction then reads the value from memory.

1. Propose a change to the 5-stage MIPS pipeline (as shown in the notes) to accommodate this addressing mode. Draw the new pipeline and explain the changes (your new design should be just as simple as the original - just five stages).
2. Give an example showing how a pipeline stall can arise with this modified design.
3. Is this design better than the original? Worse? How would you find out?

(This exercise is taken from Hennessy and Patterson).

Paul Kelly, Imperial College, 2010

332 Advanced Computer Architecture

Notes on the solutions to unassessed exercise 1

1. Propose a change to the 5-stage MIPS pipeline (as shown in the notes) to accommodate this addressing mode. Draw the new pipeline and explain the changes.

   Notice that the changes do not force us to have two ALU stages: arithmetic instructions need an ALU to do the arithmetic, but do not need to do any effective address calculation (this is why it was so important to disallow displacement addressing).

   However, in processing an arithmetic instruction with a memory operand, it is necessary to access memory *before* performing the arithmetic, so we must have the MEM cycle before the EX cycle.

   So it looks like a solution is simply to switch the MEM and EX stages in the MIPS pipeline, giving a picture like this (it's a good idea to draw it this way to show up the relative timings):

Clock:	1	2	3	4	5	6	7	8	9
Instruction 1	IF	ID	MEM	EX	WB				
Instruction 2		IF	ID	MEM	EX	WB			
Instruction 3			IF	ID	MEM	EX	WB		
Instruction 4				IF	ID	MEM	EX	WB	
Instruction 5					IF	ID	MEM	EX	WB

   The question is, can this handle all the instructions in the instruction set. For example, notice that displacement addressing has to be disallowed not only in arithmetic instructions, but also in loads and stores. *(Why?)*

2. *Give an example showing how a pipeline stall can arise with this modified design.*

   Consider the case where `R1` is used immediately after the instruction in which it is computed:

   ```
   ADD R1, R2, R3
   ADD R4, (R1), R5
   ```

   The MEM stage of the last instruction must use the value generated by the first one, but if we look at the sequence of stages in the pipeline, we see that the MEM stage for the second instruction is supposed to start in the same cycle as the EX stage in which `R1` is calculated. Thus there is no alternative but to stall.

3. *Is this design better than the original? Worse? How would you find out?*

   It's likely there would be little or no impact on clock period.

   However the number of instructions needed will be different - sometimes more, sometimes less.

   For example, consider the percentage of loads and stores which use displacement addressing (since each of these would now involve an additional instruction to compute the effective address). Of course some of these addresses could be saved in registers (we might need more registers).

What about the percentage of loads which load a value which is only used once by an ALU operation? It is in this case that the new addressing mode saves us something.

After the compilers have been changed so that displacement addressing is calculated in a register, what percentage of instructions that calculate an address are immediately followed by a load instruction which uses it? (since this is the case where a stall would occur). We would need to adjust the compiler's instruction scheduler to try to minimise this.

Paul Kelly, Imperial College, 2010