

Review of Chapters 2 and 5

- User portion of execution time is main performance metric of interest first.

-

$$\begin{aligned}\frac{sec}{prog} &= \frac{instr}{prog} \times \frac{cycles}{instr} \times \frac{secs}{cycle} \\ &= IC \times CPI \times CT\end{aligned}$$

- IC measures algorithmic complexity (when instruction mix is included it is better). It is improved by algorithmic changes.

- CT is the cycle time of the machine at it is technology dependent for a fixed architecture. Note however we have seen how it is important to decide at a digital level exactly how much combinational logic is to be allowed to settle within a cycle.
- CPI is address by architectural changes. Mainly attempting to do more instructions per cycle is the theme of later chapters.

Performance Metrics

- for a fixed program,

$$performance = \frac{1}{time}$$

so larger numbers mean better performance.

- Work per unit time metrics must be used with care.
- MIPS – what is an instruction?
- MFLOPS – better but there is still variation across multiplication, addition and division times.

- Algorithm changes can change instruction count and distribution so looking at MFLOPS for two different programs with substantially different instruction statistics for the same problem can be very misleading. For most problems of interest, when the operation count is reduced by exploiting the structure of the problem the MFLOPS metric goes down while time also goes down.
- workload characterization is crucial when benchmarking an existing system or predicting a future system's behavior.
- care must be taken when averaging performance figures across programs to compare two different machines.

Control of Datapath

- Instruction set architecture – determines the function of the instructions when applied to the programmer-visible state of the machine and their sideeffects under exceptions
- machine instructions - encoding of information from ISA in terms of operand structure classes
- Problem for Chapter 5 – given state storage (Registers, Memory), and combinational devices (ALU, MUX etc.), generate control signals to set the path of the data from some state location through combinational devices to the destination state locations that are updated and avoid incorrect update of other state elements.

Single Cycle Implementation

- Entire effect of the instruction is determined and the update prepared within one cycle
- largest amount of HW due to need to replicate function to maintain combinational form
- largest combination delay determines cycle time
- static instruction choice determines cycle time for all codes even those that do not use long instructions.

To set control lines – for each instruction:

- determine data path from source to destination through combinational devices (and their functions)
- determine information in instruction and operands needed to determine the path
- determine the setting of control lines that guarantee this data path exists and performs the correct function in the combinational logic that is adjustable (e.g. ALU).
- determine the control signals that ensure invalid data present on various lines do not update the state

- Note 2,3, and 4 yield the truth table entries for the controller combinational logic.
- Synthesize the controller combinational logic.

Multicycle Implementation

- Less HW since multiple use of same units is possible (at different times of course)
- variable instruction timings result so dynamic instruction counts for a particular program determine execution time.
- cycle time is determined by the time it takes to do basic combination operations, e.g., one pass through an ALU

To design the FSM for the controller:

- separate each instruction into a sequence of steps that satisfy the cycle time and HW constraints.
- determine the value of the control lines to perform the operations corresponding to each state (which corresponds to a step in the sequence)
- determine the information needed to define the transitions from one state to the other

- early cycles can perform potentially useful actions that may or may not be used by the particular instruction being executed, e.g., branch address calculation in state 1.
- later cycles tend to be restricted in what they do by the particular instruction executing and the data that is currently available in internal registers
- perform state reduction
- choose state encoding
- synthesize the FSM from the state/transition specifications

Exceptions and Interrupts

- source can be internal or external to a user's program
- combinational condition signals must be generated and used as input to the FSM
- FSM must have extra states to branch to the exception handler (and possibly later resume in the user's code)
- PC and cause of exception must be saved and communicated to the exception handler