Module-3
CSEN 3104
Lecture 24
03/09/2019

Dr. Debranjan Sarkar

Recapitulation till date

Thank you

Module-3
CSEN 3104
Lecture 25
16/09/2019

Dr. Debranjan Sarkar

Superscalar Architecture

What is superscalar processor?

- CPI (cycles per instruction) vs. Processor clock speed (Show diagram)
- Conventional CISC processors have CPI more than 1 (say 1 to 20)
- It is tried to lower the CPI using innovative hardware approaches
- With the use of efficient pipeline, the average CPI of RISC instructions is 1 – 2
- Superscalar processor is a subclass of RISC processors
- Superscalar design, like pipelining, is an instance of instruction-level parallelism
- Superscalar Processors allow multiple instructions to be issued simultaneously during each cycle (unlike scalar RISC which issues only 1 instructions per cycle)
- Thus the effective CPI of a superscalar processor should be lower than that of a scalar RISC processor
- The clock rate of superscalar processors matches that of scalar RISC processors

A problem on Performance of CPU

- A 50 MHz processor was used to execute a program with the following instruction mix and clock cycle counts:

Instruction type	Instruction Count	Clock Cycle Count
Integer Arithmetic	50,000	2
Data Transfer	70,000	3
Floating Point Arithmetic	25,000	1
Branch	4,000	2

- Calculate the (i) execution time of the program, (ii) effective CPI and (iii) the MIPS rating of the processor

Superscalar processor

- Scalar processors execute one instruction per cycle.
- Only one instruction is issued per cycle and only one completion of instruction is expected from the pipeline per cycle
- In superscalar processor, multiple instructions are issued per cycle and multiple results are generated per cycle
- Superscalar processors exploit more instruction-level parallelism (ILP) in user programs
- Only independent instructions can be executed in parallel without causing a wait state

Superscalar processor

- The amount of ILP varies widely depending on the type of code being executed
- It has been observed that the average value of ILP is around 2 for code without loop unrolling
- For these codes, there is not much of benefit by building a superscalar processor that can issue more than 3 instructions per cycle
- The instruction-issue degree in a superscalar processor has, therefore, been limited to 2 to 5 in practice

Pipelining in Superscalar Processor

- Instruction issue rate is the number of instructions issued per cycle, also called the degree of a superscalar processor
- The scalar base processor may be considered as a superscalar processor of degree 1
- For an m-issue superscalar machine, m instructions are issued per cycle and the ILP should be m in order to fully utilize the pipeline
- Instruction decoding, and execution resources are increased, to form effectively m pipelines operating concurrently
- At some pipeline stages, the functional units may be shared by multiple pipelines
- Show figure of a dual-pipeline superscalar processor
- Here the processor can issue two instructions per cycle, if there is no resource conflict and no data dependence problem

Pipelining in Superscalar Processor

- There are essentially two pipelines, both having 4 processing stages viz. fetch, decode, execute and store
- Each pipeline has its own fetch unit, decode unit, and store unit
- The two instruction streams, flowing through the two pipelines, are retrieved from a single source stream (I-cache)
- For simplicity, we may assume that each pipeline stage requires one cycle, except the execute stage which may require a variable number of cycles
- Using multiple functional units one can avoid structural hazards
- This concept is used very effectively in constructing superscalar processors
- Four functional units viz. multiplier, adder, logic unit and load unit are available for use in the execute stage
- These functional units are shared by two pipelines on a dynamic basis
- The multiplier itself has 3 pipeline stages, the adder has 2 stages and the others have only 1 stage each

Pipelining in Superscalar Processor

- The two store units (S1 and S2) can be dynamically used by the two pipelines, depending on availability at a particular cycle
- There is a lookahead window with its own fetch and decoding logic
- This window is used for instruction lookahead in case out-of-order instruction issue is desired to achieve better pipeline throughput
- It requires complex logic to schedule multiple pipelines simultaneously, especially as the instructions are retrieved from the same source
- The aim is to avoid pipeline stalling and minimize pipeline idle time

Superscalar performance

- Relative performance of a superscalar processor with that of a base scalar machine
- Let there be N independent instructions through the k-stage pipeline
- We estimate the ideal execution time in both the cases
- Time required by the base scalar machine is

  T_{bs} = 1.k + (N - 1).1 (cycles) = k + N – 1 cycles
- Let us consider an m-issue superscalar machine
- Time required by the superscalar machine is

  T_{ss} = k + (N - m)/m (cycles)
- As the first m instructions are executed in k cycles through the m pipelines simultaneously and for the remaining (N-m) instructions, the rate at which the instructions are executed is m per cycle
- So, the ideal speedup of the superscalar machine over the base scalar machine is

  $T_{ss} / T_{bs} = m(N + k – 1)/ (N + m(k - 1))$
- As N → ∞, the speedup limit tends to m

Thank you

Module-3
CSEN 3104
Lecture 26
17/09/2019

Dr. Debranjan Sarkar

Superscalar Architecture

Data Dependences

- Show the example program and the dependence graph
- Register R1 is loaded by I1 and its content is used by I2, so we have flow dependence: I1 → I2
- The result in register R4 after executing I4 may affect the operand register R4 used by I3, we have anti-dependence between I3 and I4
- Since both I5 and I6 modify the register R6, and R6 supplies an operand for I6, we have both flow and output dependence between I5 and I6
- These are shown in the dependence graph
- To schedule instructions through one or more pipelines, these data dependences must not be violated. Otherwise erroneous results may be produced

Pipeline stalling

- Pipeline can be stalled due to
  - data dependences, or
  - by resource conflicts among instructions
    - already in the pipeline, or
    - about to enter the pipeline
- This problem may seriously lower pipeline utilization
- This problem exists in both scalar and superscalar processors
- It is more serious in a superscalar pipeline
- Proper scheduling avoids pipeline stalling

Pipeline Stalling

- Show figures to give examples
- Figure (a) shows the case of no data dependences
  - All pipeline stages are utilized without idling
- Figure (b) shows the case of flow dependences (I1 → I2)
  - I2 in the second pipeline must wait for 2 cycles, before entering the execution stages
  - This delay may also pass to the next instruction I4 entering the pipeline
- Figure (c) shows the effect of branching (instruction I2)
  - A delay slot of 4 cycles results from a branch taken by I2 at cycle 5
  - Therefore, both pipelines must be flushed before I3 and I4 can enter the pipelines from cycle 6
  - Here delayed branch and other amending actions are not taken

Pipeline Stalling

- Figure (d) shows the case of no resource conflicts
  - All pipeline stages are utilized without idling
- Figure (e) shows a combined problem of both resource conflict and data dependences (I1 and I2 use the same functional unit, and I2 → I4)
  - I2 must be scheduled 1 cycle behind because the two pipeline stages (e1 and e2) of the same functional unit must be used by I1 and I2 in an overlapped fashion
  - For the same reason, I3 is also delayed by 1 cycle
  - I4 is delayed by 2 cycles due to the flow dependence on I2

A dual-pipeline, superscalar processor with 4 functional units and a lookahead window.

Superscalar pipeline scheduling

- Instruction issue and completion policies are critical to superscalar processor performance
- Three scheduling policies:
  - In-order issue (Program order not violated. Easy to implement but may not yield the optimal performance)
    - In-order completion
    - Out-of-order completion
  - Out-of-order issue (Program order violated. Purpose is to improve performance)
    - Usually ends up with Out-of-order completion

Superscalar pipeline scheduling (In-order issue)

- Six instructions are issued in program order
- As I1→I2 (flow dependent), I2 has to wait one cycle to use the data loaded in by I1
- I3 is delayed by 1 cycle because the same adder a1 is being used by I2
- I6 has to wait for the result of I5 before it can enter the multiplier stages
- To have in-order completion, I5 has to wait for 2 cycles to come out of pipeline 1
- In total, 9 cycles are needed and 5 idles are observed
- If out-of-order completion is allowed, I5 is allowed to complete ahead of I3 and I4, which are totally independent of I5
- The total execution time does not improve but the pipeline utilization rate does
- Only 3 idle cycles are observed
- Show the figures of the scheduling policy In-order Issue

Thank you

Module-3
CSEN 3104
Lecture 27

Dr. Debranjan Sarkar

Superscalar Architecture

Superscalar pipeline scheduling (Out-of-order issue)

- Lookahead window is used to reorder the instruction issues in order to shorten the total execution time
- By using the lookahead window, I5 can be decoded in advance because it is independent of all the other instructions
- I5 is fetched and decoded by the window, while I3 and I4 are decoded concurrently
- It is followed by issuing I6 and I1 at cycle 2, and I2 at cycle 3
- As the issue is out of order, the completion is also out of order
- Total execution time is reduced to 7 cycles with no idle stages
- Show the figures of the scheduling policy Out-of-order Issue

Superscalar pipeline scheduling

- In-order issue and completion is the simplest one to implement
- Unnecessary delays happen to maintain the program order
- So it is rarely used even in a conventional scalar processor
- In a multi-processor environment, this policy is attractive
- Allowing out-of-order completion can be found in both scalar and superscalar processors
- Some long-latency operations (e.g. Load and Floating point operations) can be hidden in out-of-order completion to achieve a better performance
- Output dependence and anti-dependence are the two relations that prevent out-of-order completion
- Out-of-order issue gives the processor more freedom to exploit parallelism, and thus pipeline efficiency is enhanced

Static Arithmetic Pipelines

- Arithmetic pipelines are mostly designed to perform fixed functions
- ALUs perform fixed-point and floating-point operations separately
- Fixed-point unit is also called the Integer Unit
- Floating-point unit may be built either as part of the CPU or on a separate coprocessor

Arithmetic Pipeline Stages

- Add, subtract, multiply, divide, squaring, square rooting, logarithm etc can be implemented with the basic add and shift operations
- A typical 3-stage floating-point adder includes
  - a 1st stage for exponent comparison and equalization (adder + shifting logic)
  - a 2nd stage for fraction addition (a high-speed carry look-ahead adder)
  - a 3rd stage for fraction normalization and exponent readjustment (shifter + adder)

Arithmetic Pipeline Stages

- Arithmetic or logical shifts can be implemented with shift registers
- High speed addition requires
  - Carry-propagation adder (CPA), or Show figure
  - Carry-save adder (CSA) Show figure
- In CPA, the carries propagate from the low end to the high end, using
  - Ripple carry propagation, or
  - Some carry look-ahead technique

Arithmetic Pipeline Stages

- In CSA, the carries are not allowed to propagate, but are saved in a carry vector
- Let X, Y, and Z are three n-bit input numbers, expressed as

$X = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ $Y = (y_{n-1}, y_{n-2}, \ldots, y_1, y_0)$ $Z = (z_{n-1}, z_{n-2}, \ldots, z_1, z_0)$

- The CSA performs bitwise operations simultaneously to produce two n-bit output numbers, denoted as

  $S^b = (0, S_{n-1}, S_{n-2}, \ldots, S_1, S_0)$ and

  $C = (C_n, C_{n-1}, \ldots, C_1, 0)$

- The input-output relationships (for i = 0, 1, 2, ..., n-1) are expressed as:

  $S_i = x_i$ XOR y_i XOR z_i

  $C_{i+1} = x_i y_i$ OR $y_i z_i$ OR $z_i x_i$

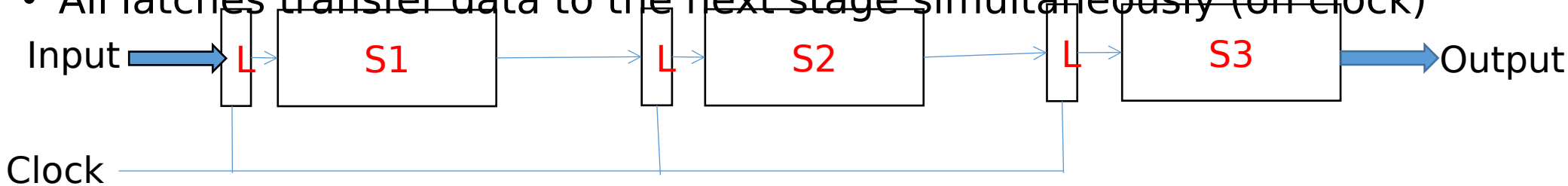
- $S = X + Y + Z = S^b + C$ (using CPA)

Multiply Pipeline Design

- Example of multiplication of two 8-bit integers A X B = P
- P is the 16-bit product
- Fixed point multiplication is the summation of 8 partial products (Show example)
- $P = A \times B = P_0 + P_1 + P_2 + P_3 + P_4 + P_5 + P_6 + P_7$
- P_j is (8+j) bits long with j trailing zeros
- The summation of 8 partial products is done with a Wallace tree of CSAs plus a CPA at the final stage (Show figure)
- S1: generates eight partial products
- S2: two levels of 4 CSAs taking eight numbers and producing four
- S3: two CSAs convert four numbers into two numbers
- S4: one CPA takes two numbers and result into one number

Principles of Pipelining (Recapitulation)

- Decomposes a sequential task into subtasks
- Each subtask is executed in a special dedicated stage
- These stages are connected with one another to form a pipe like structure.
- Instructions enter from one end and exit from another end
- Stages are pure combinational circuits for arithmetic or logic operations
- Result obtained from a stage is transferred to the next stage
- Final result is obtained after the instruction has passed through all the stages
- Stages are separated by high speed latches (or registers)
- All latches transfer data to the next stage simultaneously (on clock)



Multiply Pipeline Design

- For a maximum width of 16 bits, the CPA needs 4 gate levels of delay
- Each level of the CSA can be implemented with a 2 gate-level logic
- The delay of the first stage (S_1) also involves 2 gate levels
- So, all pipelines stages have an approximately equal amount of delay
- The matching of stage delays is important to determine the number of pipeline stages and the clock period
- If the delay of the CPA stage can be reduced to match

Thank you