Computer Organization &
Assembly Language Programming

CSE 2312
Lecture 14 IA-64

Junzhou Huang, Ph.D.
Department of Computer Science and Engineering
Reviewing (1): Flow Control

• What?
  – Flow of control refers to the sequence in which instructions are executed dynamically during program execution.

• Related Factors
  – Successively-executed instructions are fetched from consecutive memory locations, in the absence of branches and procedure calls
  – Procedure calls cause the flow of control to be altered, stopping the procedure currently executing and starting the called procedure.
  – Coroutines are related to procedures and cause similar alterations in the flow of control. They are useful for simulating parallel processes.
  – Traps and interrupts also cause the flow of control to be altered when special conditions occur.
Reviewing (2): Sequential Flow of Control and Branches

• Most instructions do not alter the flow of control.
  – After each instruction, the program counter is increased by the instruction length.
  – If observed over an interval of time that is long compared to the average instruction time, the program counter is approximately a linear function of time

• Containing Branches
  – This relation is not true
  – Program Counter is no longer a monotonically increasing function of time
  – It becomes difficult to visualize the instruction execution sequence from the program listing.

Program counter as a function of time (smoothed)
(a) Without branches. (b) With branches.
Reviewing (3): Procedures

• What?
  – The most important technique for structuring programs

• Procedures and Flow of Control
  – A procedure call alters the flow of control just as a branch does
  – Unlike the branch, when finished performing its task, it returns control to the statement or instruction following the call.
  – A procedure body can be regarded as defining a new instruction on a higher level.
  – Procedure call can be thought of as a single instruction, even though the procedure may be quite complicated.
  – To understand a piece of code containing a procedure call, it is only necessary to know what it does, not how it does it.

• Recursive Procedure
  – A procedure that calls itself, either directly or indirectly via a chain of other procedures.
  – Tower of Hanoi: an example of a recursive procedure.
Reviewing (4): Recursive Procedures

- **Towers of Hanoi**
  - There are three gold pegs. Around the first one were a series of 64 concentric gold disks, each with a hole in the middle for the peg.
  - Each disk is slightly smaller in diameter than the disk directly below it. The second and third pegs were initially empty.
  - How to transfer all the disks to peg 3, one disk at a time, but at no time may a larger disk rest on a smaller one?
Reviewing (5): Calling/Called Procedures

• Distinction
  – Procedure A calls a procedure B
  – Symmetric situation: neither A nor B is a main program, both being procedures; first control is transferred from A to B—the call—and later control is transferred from B to A—the return.
  – Asymmetry: 1) when control passes from A to B, procedure B begins executing at the beginning; when B returns to A, execution starts not at the beginning of A but at the statement following the call.
  – If A calls B many times, B starts at the beginning all over again each and every time, whereas A never starts over again.
Reviewing (6): Coroutines

• **What?**
  – Two procedures, each of which calls the other as a procedure.
  – When B returns to A, it branches to the statement following the call to B.
  – When A transfers control to B, it does not go to the beginning (except the first time) but to the statement following the most recent “return”
  – Two procedures that work this way are called coroutines.

• **Usage?**
  – Simulate parallel processing on a single CPU. Each coroutine runs in pseudo-parallel with the others, as though it had its own CPU.
  – It also is useful for testing software that will later actually run on a multiprocessor.

When a coroutine is resumed, execution begins at the statement where it left off the previous time, not at the beginning.
Reviewing (7): Trap

• Trap
  – An automatic procedure call initiated by some condition caused by the program, usually an important but rarely occurring condition.
  – A good example is overflow. If the result of an arithmetic operation exceeds the largest number that can be represented, a trap occurs to switch the flow of control to some fixed memory location instead of continuing in sequence.
  – At that fixed location is a branch to a procedure called the trap handler, which performs some appropriate action, such as printing an error message.
  – If the result of an operation is within range, no trap occurs.

• Common conditions that can cause traps
  – floating-point overflow, floating-point underflow, integer overflow
  – protection violation, undefined opcode, stack overflow,
  – attempt to start nonexistent I/O device,
  – attempt to fetch a word from an odd-numbered address
  – division by zero.
Reviewing (8): Interrupt

• **Interrupt**
  – Interrupts are changes in the flow of control caused not by the running program, but by something else, usually related to I/O.
  – When finished, the interrupt handler returns control to the interrupted program.
  – It must restart the interrupted process in exactly the same state that it was in when the interrupt occurred, which means restoring all the internal registers to their preinterrupt state.

• **Trap vs. Interrupt**
  – Traps are synchronous with the program and interrupts are asynchronous.
  – If the program is rerun a million times with the same input, the traps will reoccur in the same place each time but the interrupts may vary, depending, for example, user I/O operations.
  – The reason for the reproducibility of traps and irreproducibility of interrupts is that traps are caused directly by the program and interrupts are indirectly caused by the program.
Reviewing (9): Interrupt Priority

- **Interrupt Priority**
  - Assign each I/O device a priority, high for very critical devices and low for less critical devices.
  - Similarly, the CPU should also have priorities, typically determined by a field in the PSW.
  - When a priority n device interrupts, the interrupt routine should also run at priority n.
  - While a priority n interrupt routine is running, any attempt by a device with a lower priority to cause an interrupt is ignored until the interrupt routine is finished and the CPU goes back to running lower priority code.
  - On the other hand, interrupts from higher-priority devices should be allowed to happen with no delay.
IA 64 and Itanium 2

• IA-64
  – Speeding up the implementation of IA-32 ISA is getting harder and harder as the constraints imposed by the IA-32 ISA
  – The only real solution is to abandon the IA-32 and go to a completely new ISA.
  – The EMT-64 is a wider version of the Pentium 4, with 64-bit registers and a 64-bit address space. This processor solves the address space problem but still has the implementation complexities of the Pentium 4.
  – IA-64, developed jointly by Intel and Hewlett Packard, is a full 64-bit machine, not an extension of an existing 32-bit machine.
Pentium 4

- Problem 1-3
  - IA-32 is an ancient ISA with all the wrong properties for current technology. It is a CISC ISA with variable-length instructions and a myriad of different formats that are hard to decode quickly on the fly. Current technology works best with RISC ISAs that have one instruction length and a fixed-length opcode that is easy to decode.
  - The IA-32 is a two-address memory-oriented ISA. Current technology favors load/store ISAs that only reference memory to get the operands into registers but otherwise perform all their calculations using three-address memory register instructions.
  - The IA-32 has a small and irregular register set. This requires intermediate results to be spilled into memory all the time, generating extra memory references even when they are not logically needed.
Pentium 4

- Problem 4-6
  - The small number of registers causes many dependences because results have to go somewhere and there are no extra registers available. To avoid blocking on cache misses too often, instructions have to be executed out of order. These require a lot of very complex hardware.
  - Doing all this work quickly requires a deep pipeline. In turn, the deep pipeline means that instructions are entered into it take many cycles before they are finished. Consequently, very accurate branch prediction is essential to make sure the right instructions are being entered into the pipeline. Because a misprediction requires the pipeline to be flushed, at great cost, even a fairly low misprediction rate can cause a substantial
  - To alleviate the problems with mispredictions, the processor has to do speculative execution, with all the problems that entails, especially when memory references on the wrong path cause an exception.
IA-64

• **Key Idea**
  – Moving work from run time to compile time.
  – Pentium 4 does a lot of other work to determine how to keep all the hardware resources fully occupied.
  – In the IA-64 model, the compiler figures out all these things in advance and produces a program that can be run as is, without the hardware having to juggle everything during execution.
  – The model of making the underlying parallelism in the hardware visible to the compiler is called RPIC (Explicitly Parallel Instruction Computing).

• **Important Features**
  – Reducing memory references
  – Instruction scheduling
  – Reducing conditional branches
  – Speculation
Reducing Memory References

- **Avoiding Memory Access as possible**
  - Each procedure has access to 32 static registers and some (variable) number of dynamically allocated registers.
  - 128 floating registers, 128 special purpose registers,
  - 64 1-bit predicate register, 8 Branch registers
Instruction Scheduling

An IA-64 bundle contains three instructions.

- **Scheduling Instructions**
  - One of the main problems in the Pentium 4 is the difficulty of scheduling the various instructions over the various functional units and avoiding dependences.
  - IA-64 and Itanium 2 avoid these problems by having the compiler do these.
  - The key idea is that a program consists of a sequence of instruction groups.
Reducing Conditional Branches: Predication (1)

(a) An if statement. (b) Generic assembly code for a). (c) A conditional instruction.

- **Condition Execution**
  - In (c), we get rid of the conditional branch by using a new instruction, CMOVZ, which is a conditional move.
  - What it does is check to see if the third register, R1, is 0.
  - If so, it copies R3 to R2. If not, it does nothing.
Reducing Conditional Branches: Predication (2)

(a) An if statement.  (b) Generic assembly code for a). (c) A conditional instruction.

- **Condition Execution**
  - The CMOVNs and the CMOVZs form a single basic block with no conditional branch.
  - IF part: CMOVZ
  - ELSE part: CMOVN
Reducing Conditional Branches: Predication (3)

(a) An if statement. (b) Generic assembly code for a). (c) A conditional instruction.

- **Condition Execution**
  - The CMPEQ instruction compares two registers and sets the predicate register P4 to 1 if they are equal and to 0 if they are different.
  - It also sets a paired register, say, P5, to the inverse condition.
  - Now the instructions for the if and then parts can be put after one another, each one conditioned on some predicate register (shown in angle brackets).
Speculative Loads

• **What?**
  – If a LOAD is speculative and it fails, instead of causing an exception, it just stops and a bit associated with the register to be loaded is set marking the register as invalid.

• **Usage**
  – The way speculation is normally used is for the compiler to hoist LOADs to positions before they are needed.
  – By starting early, they may be finished before the results are needed.
  – At the place where the compiler needs to use the register just loaded, it inserts a CHECK instruction.
  – If the value is there, CHECK acts like a NOP and execution continues immediately.
  – If the value is not there yet, the next instruction must stall.
  – If an exception occurred and the poison bit is on, the pending exception occurs at that point.
IA-64 Summary

• A machine implementing the IA-64 architecture gets its speed from several sources:
  – At the core is a state-of-the-art pipelined, load/store, three address RISC engine.
  – In addition, IA-64 has a model of explicit parallelism that requires the compiler to figure out which instructions can be executed at the same time without conflicts and group them together in bundles. In this way the CPU can just blindly schedule a bundle without having to do any heavy thinking.
  – Next, predication allows the statements in both branches of an if statement to be merged together in a single stream, eliminating the conditional branch and thus the prediction of which way it will go.
  – Finally, speculative LOADs make it possible to fetch operands in advance, without penalty if it turns out later that they are not needed after all.