CS M151B:
Computer Systems Architecture

Week 3

Adder Progress

- Ripple Carry Adder:
  - Too slow, 160T delay for last result (Cout_31) based fan-out metric.
  - Problem: Each adder relies on the computation of previous adder's Cout
- Pure Carry Lookahead Adder:
  - Redundant calculations based on signals that are ready at time 0.
  - Better performance: 68T delay for last result.
  - Problem: Too large/impractical, fan-out of 32 (!)

Adder Progress

- Partial Carry Lookahead:
  - Create small CLA modules and chain them together, ripple style.
  - Better use of space, slower than full Carry Lookahead based on the current metric used (fan-in of n <=> n*T delay).
- Can we do better?
  - Like, waaay waaay better?

Hierarchical CLA

- Notice that each CLA unit consists of a Cin, two 4-bit inputs, and a Cout.
- This is like a 4-bit full adder!
- When combine via ripple style, each CLA unit must wait for previous CLA's Cout, which causes a delay.
- Let's combine these CLA units in CLA style.
- Why does this help?

Terminology

- Cn = Cin to unit n
- Sn = Sum out of unit n
- Gn = Generate of unit n (Ga, Gb, etc.)
- Pn = Propagate of unit n
- Delay of gate with fan-in of n is n*T (this is not always true!)

Note

- The following slides are to be used in conjunction with the lecture 3 slides, pg. 23-28.

Hierarchical CLA

- Recall that the S3 and C4 of a 4-bit CLA unit has delay 12T.
- We don't want to wait that long for the second CLA unit to be ready.
- We would like the Cin to each CLA unit to be based on C0 rather than C4, C8, C12, etc.

Hierarchical CLA

- We do this by creating a Propagate and Generate signal for each CLA unit.
- Ga = G0*P1*P2*P3 + G1*P2*P3 + G2*P3 + G3
- Pa = P0*P1*P2*P3
- The time for each Gx is 10T and the time for each Px is 6.
- C4 = Ga + C0 * Pa
  - C4 ready at 12T. This isn't any different from before.

Hierarchical CLA

- C8 = Gb + Ga * Pb + C0 * Pa * Pb
  - C8 ready at 15T. This IS different.
- C12 = Gc + Gb *Pc + Ga * Pb * Pc + C0 * Pa * Pb * Pc
  - C12 ready at 17T
- ...and so on.
- C32 (or cout_31) is 27 T (!)
- Is this the max latency from this module?

Hierarchical CLA

- No, we still need S31, which is still defined by: S31 = (A31 ^ B31) ^ C31, but remember that this is also a CLA module which means that C31 is actually created as C31 = G30 + G29 * P30 + … + C28 * P28 * P29 * P30
- C28 is ready at 25T
- C31 is ready at 33T
- S31 is ready at 35T
- This is a very good result, but this will require 8 CLA units together, which means the last values will require fan-in of 8.

Hierarchical CLA

- Let's add one last abstraction
- Rather than:
  - A single level-two hierarchical CLA with 8 level-one CLAs,
- Let's have:
  - Two level-two hierarchical CLA's with 4 level-one CLAs, chained together ripple style.

Hierarchical + Ripple CLA

- Out of first hierarchical CLA, C16 is 19T.
- Now from second hierarchical CLA, Cin is 19T.
- C20 = Ga + C16 * Pa
- Etc..
- Note that with the previous Full Hierarchical, the latency came from the term Ga*Pd*Pc*Pb rather than C0*Pd*Pc*Pb*Pa
- C32 = 29T
- What about S31?

Hierarchical + Ripple CLA

- S31 = (A31 ^ B31) ^ C31
- C31 = G30 + G29 * P30 + … + C28 * P28 * P29 * P30
- Except now, C28 is 27T
- S31 = 37T
- Recall that the original Ripple Carry was around 160T and even the full CLA was around 65T. Be amazed!
- Had enough yet?

No you haven't

- Let's take a step back. The Ripple Carry adder would still be the most convenient if not for the fact that you had to cascade carries.
- Ideally, we want all results to be completed simultaneously, but this is impossible.
- However, perhaps all of the calculations can be done simultaneously.

Carry Select Adder

- Let's make a Ripple Carry Adder, except instead of waiting for the previous adder's carry to be done to use it, for each adder, do both calculations as if carry in is 0 or 1.
- This requires two adders per bit.
- Now, once the carry out is calculated, simply tell the next unit which results to use.
- Cout = (A & B) | (B & Cin) | (A & Cin)
- S = (A ^ B) ^ Cin

Carry Select Adder

- Cout = (A & B) | (B & Cin) | (A & Cin)
- S = (A ^ B) ^ Cin
- Cout is ready is 5T
- S is ready in 4T
- Each adder has the possible results ready in this time. The Cout must simply propagate to tell all of the other units.
- Each propagation now only adds cost of choosing which result to use (muxing)

Carry Select Adder

- C1 is ready in 5T
- S0 is ready in 4T
- C2 is ready in 5T + M
- S1 is ready in 4T + M
- C3 is ready in 5T + 2M
- …
- C32 is ready in 5T + 31M
- S31 is ready in 4T + 31M

Carry Select Adder

- M is probably pretty small, but having 31 M's is going to add up.
- We turn to our good friend the CLA
- Instead, chain eight, 4-bit CLA's with the same concept of doubling up on calculations.

Carry Select Adder

- C4 in 4-bit CLA is 12T
- S3 in 4-bit CLA is 12T
- C8 is ready in 12T + M
- S7 is ready in 12T + M
- …
- C32 is ready in 12T + 7M
- S31 is ready in 12T + 7M
- Are we done yet?

Tradeoffs

- What were the tradeoffs that have driven this process?

Tradeoffs

- What were the tradeoffs that have driven this process?
  - Time
  - Space (fan-out)
  - Space (redundant gates to do redundant computation)

Tradeoffs

- The Ripple Carry was the smallest and simplest, but slowest.
- How did full CLA improve upon the Ripple Carry?

Tradeoffs

- The Ripple Carry was the smallest and simplest, but slowest.
- How did full CLA improve upon the Ripple Carry?
  - Redundant operations to deal strictly with inputs that were ready at time 0.
- How did the Partial Carry improve upon or worsen compared to full CLA?

Tradeoffs

- How did the Partial Carry improve upon or worsen compared to full CLA?
  - Full CLA had very wide, operations with very low depth. (P31*P30*P29*...*G0)
  - Reduce the width of operations but increase the depth. Ultimately more operations and a bit slower. Note: This may not be true in the real world as a gate with a fan-in of 32 may actually incur a very significant delay.
- How did the single Hierarchical CLA improve upon the Full and Partial CLA?

Tradeoffs

- How did the Hierarchical CLA improve upon the Full CLA?
  - Add even more redundant computations, doing computations in different order.
- How did the Carry Select improve (depending on MUX time) upon the Hierarchical CLA?

Tradeoffs

- How did the Carry Select improve (depending on MUX time) upon the Hierarchical CLA?
  - Full redundancy with respect to the one unpredictable factor (carry in). Doubled all of the hardware.
- How do we improve upon Carry Select?

Midterm Review: Performance

- ET = CPI * IC * CT
- Assume for a given processor the following stats:

	Arith	Load/Store	Branch
Cycles	1	10	3
No. Insns	500 * 10^6	300 * 10^6	100 * 10^6

Midterm Review: Performance

- Suppose we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

Midterm Review: Performance

- Old:
  - IC = 900*10^6
  - CPI = 5/9 * 1 + 3/9 * 10 + 1/9 * 3 = 38/9
  - ETo = 38/9 * 900 * 10^6 * CT = 3800 * 10^6 * CT
- Improve arith by 2:
  - IC = SAME
  - CPI = 5/9 * ½ + 3/9 * 10 + 1/9 * 3 = 71/18
  - ETn = 3550 * 10^6 * CT
- “Speedup”?

Midterm Review: Performance

- Old:
  - ETo = 3800 * 10^6 * CT
- Double:
  - ETn = 3550 * 10^6 * CT
- What percent did we reduce original time by?
- How many times faster is the new time?

Midterm Review: Performance

- Old:
  - ETo = 3800 * 10^6 * CT
- Double:
  - ETn = 3550 * 10^6 * CT
- What percent did we reduce original time by?
  - ETo * (1 – n) = ETn
  - n = .0658
- How many times faster is Etn?
  - ETo/ETn = 1.0704

Midterm Review: Performance

- Old:
  - ETo = 3800 * 10^6 * CT
- Improve arith by 10:
  - ETn = 3350 * 10^6 * CT
- What percent did we reduce original time by?
  - ETo * (1 – n) = ETn
  - n = .1184
- How many times faster is ETn?
  - ETo/ETn = 1.1343 <=> NOT MUCH BETTER, AMDAHL STRIKES AGAIN!

Midterm Review: Tradeoffs

- In general, what can affect CPI?

Midterm Review: Tradeoffs

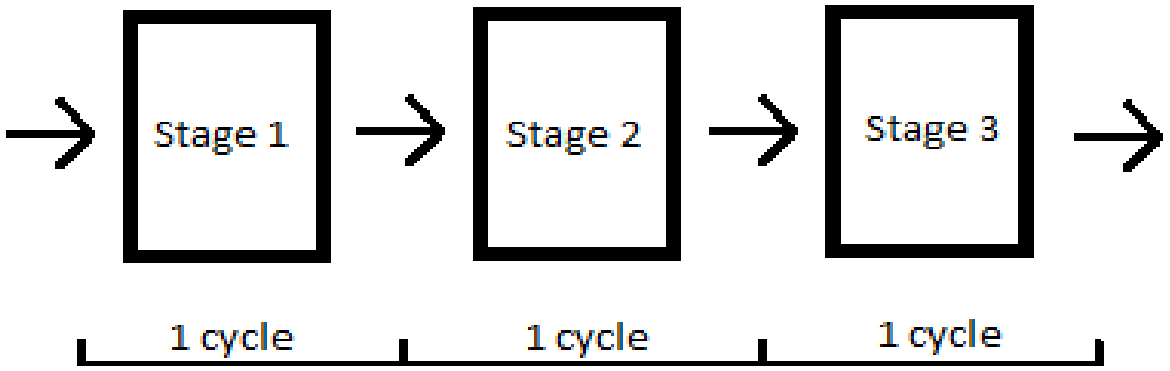
- In general, what can affect CPI?
  - Change how the program is written to use different proportions of instruction types (recall CPI is average over entire program)
  - Change how the program is compiled to use different proportions blah blah blah.
  - The types of instructions that the ISA defines: more complicated ones may mean more CPI and simpler ones may mean fewer CPI.

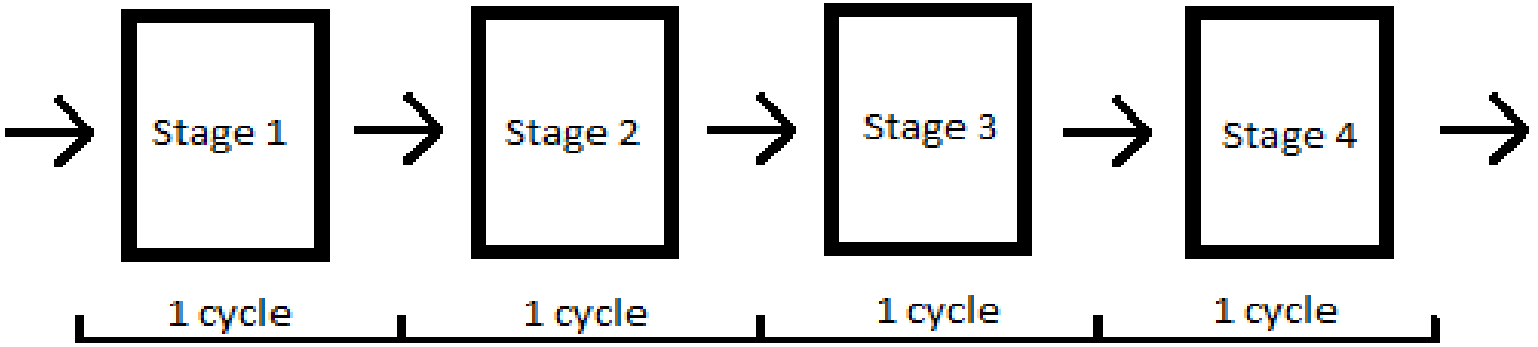
Midterm Review: Tradeoffs

- In general, what can affect CPI?
  - Change the complexity of hardware (ie adding/removing/splitting modules)

Midterm Review: Tradeoffs

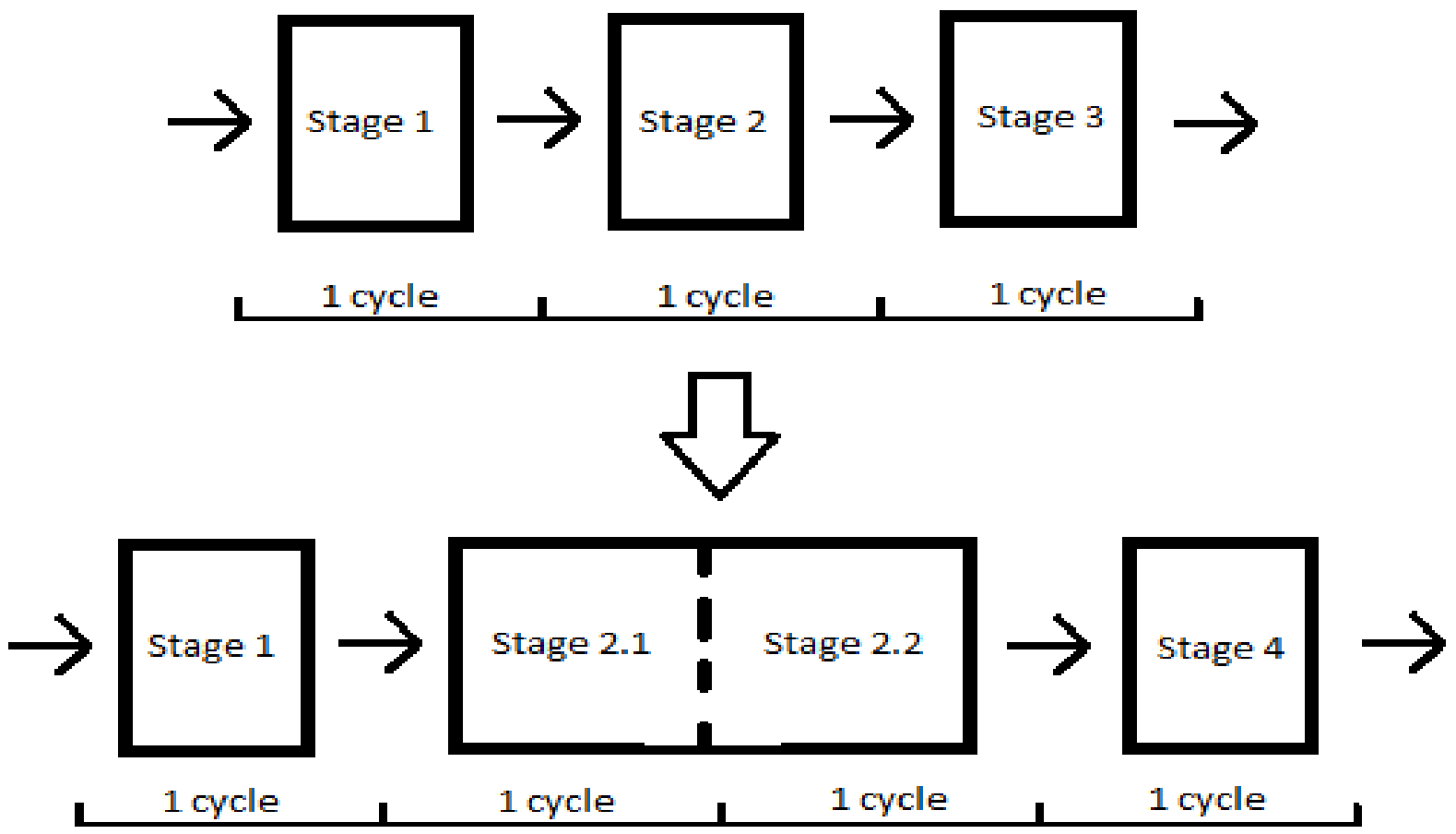
- Add module:





Midterm Review: Tradeoffs

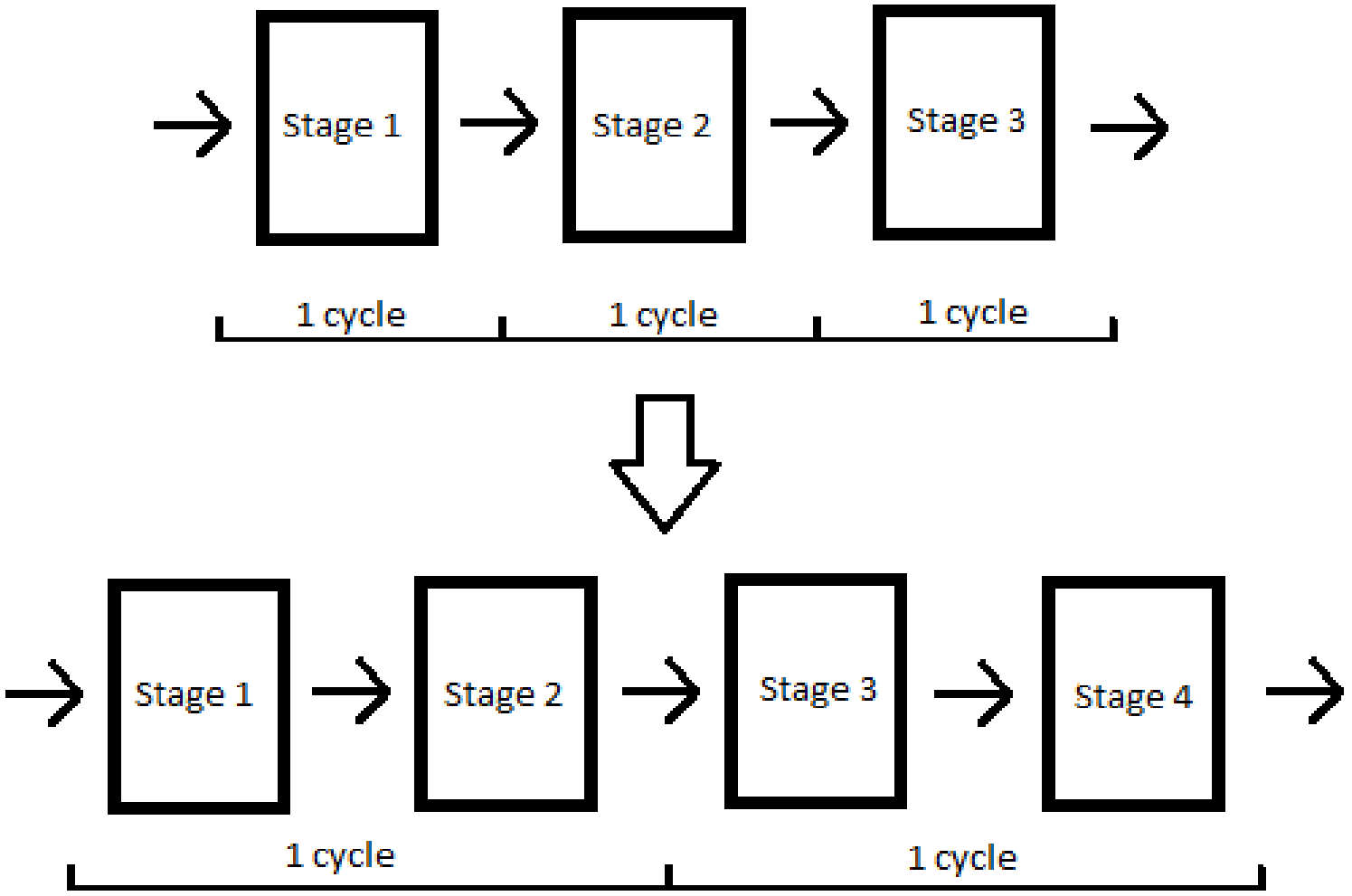
- Split module:



- Why might we do this?
- Could we add a module but decrease CPI?

Midterm Review Tradeoffs

- Sure:



- What else have we changed now?

Midterm Review: Tradeoffs

- In general, what can affect CT?

Midterm Review: Tradeoffs

- In general, what can affect CT?
  - Mostly microarchitecture stuff.
  - Change CT so that multiple modules are expected to complete in one clock cycle.
  - Split a one cycle module into two modules, each of which takes one cycle.
  - Will splitting always change the clock time?

Midterm Review: Tradeoffs

- Will splitting always decrease the clock time?
  - No. If original module was the bottleneck, then the CT will decrease.
  - However, if the original module was not the bottleneck, then the CT cannot be changed since the bottleneck lies elsewhere and there is still a module that expects to complete in one clock cycle.

Midterm Review: Tradeoffs

- In general, what can affect IC?
  - Program written/compiled differently to choose more or fewer instructions.
  - The type of instructions that the ISA defines. If ISA defines many complex instructions IC may go down. If ISA defines only simple instructions, IC may go up.

Midterm Review: Tradeoffs

- An example mentioned in class:
- Consider J-Type instructions:
- [6 bits][26 bits]
- opcode addr
- What if we want to change it so that we only use 25 bits for addr. What affect could this have?

Midterm Review: Tradeoffs

- New format (assume fixed-length):
- [7 bits][25 bits]
- opcode addr
- What are the direct effects?

Midterm Review: Tradeoffs

- 6 bit opcode → 7 bit opcode
- More operations:
  - 64 j-type instructions → 128 j-type instructions
- 26 bit addr → 25 bit addr
- Lower jump range:
  - [PC + 4]31-28 : ADDR * 4 → [PC + 4]31-27 : ADDR * 4
- What indirect effects could these direct effects have?

Midterm Review: Tradeoffs

- IC?

Midterm Review: Tradeoffs

- IC?
  - Increase: need to resort to direct jumping more often (load address into register, then use jr)
  - Decrease: more instructions to choose from, may simplify other operations

Midterm Review: Tradeoffs

- CPI?

Midterm Review: Tradeoffs

- CPI?
- Do the new instructions increase the complexity of the microarchitecture? If yes, then CPI may have increased.
- The cost of resorting to use add + jr instead may have changed proportion of instruction types, changing CPI.
- Using the new instructions could have changed the proportion of instruction types, changing CPI.

Midterm Review

- CT?

Midterm Review

- CT?
- In what way has the microarchitecture changed?
- If we split a bottleneck into two cycles, the CT may have decreased.
- If we simply make the a module more complex, we may increase the CT to accommodate.

End of

The Third Week

-Seven Weeks Remain-