Registers are just arrows

Discussion exercise

How can we design our ISA without registers?

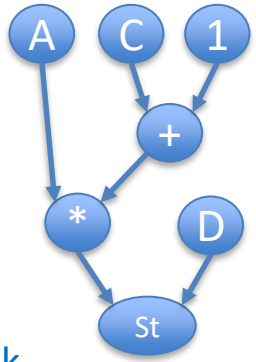
Can we simplify the microarchitecture as a result?

Perhaps encoding dependence directly (so it doesn't have to be discovered)

Perhaps avoiding register renaming?

Code is just an encoding for a graph

```
Load r1 A
Load r2 C
Add r3 r2 #1 // r3 = C+1
Mul r4 r1 r3 // r4 = r3*A
Store r4 D   // D = r4
```



Register machine

```
Push A    // load from location A, push onto stack
Push B
Add #1    // add 1 to the value on the top of the stack
Mul       // multiply the top two items on the stack
Store D   // store the value on the top of the stack to memory
```

Stack machine

```
Load +3 A  // load from location A, send to Mul instruction below
Load +1 C  // load from location C, send to Add instruction below
Add +1 #1 // add 1 to the value received, send result to next
Mul +1     // multiply value received by from Add by C, send to store
Store D    // store the value received from instruction above
```

dataflow machine

```
Load A     // Load A, drop the value on the conveyor "belt"
Load C     // Load C, drop the value on the conveyor "belt"
Add -1 #1 // add 1 to the result of the instruction one instruction earlier
Mul -1 -3  // multiply the results from positions -1 and -3 on the belt
Store -1 D // store the result of the multiply to memory
```

"belt" machine

Dataflow instruction sets

- Examples: EDGE ("Explicit Data Graph Execution), TRIPS, Wavescalar

  (and earlier historical designs – Manchester Dataflow, MIT Dataflow etc)

- Objectives:
  - More instructions per cycle
  - Larger instruction "window"
  - Improved energy efficiency

Bundle instructions into statically-scheduled dataflow fragments

O-O-O at the fragment level

Window capacity is counted in fragments

O-O-O "Turing Tax" is paid per fragment, not per-instruction

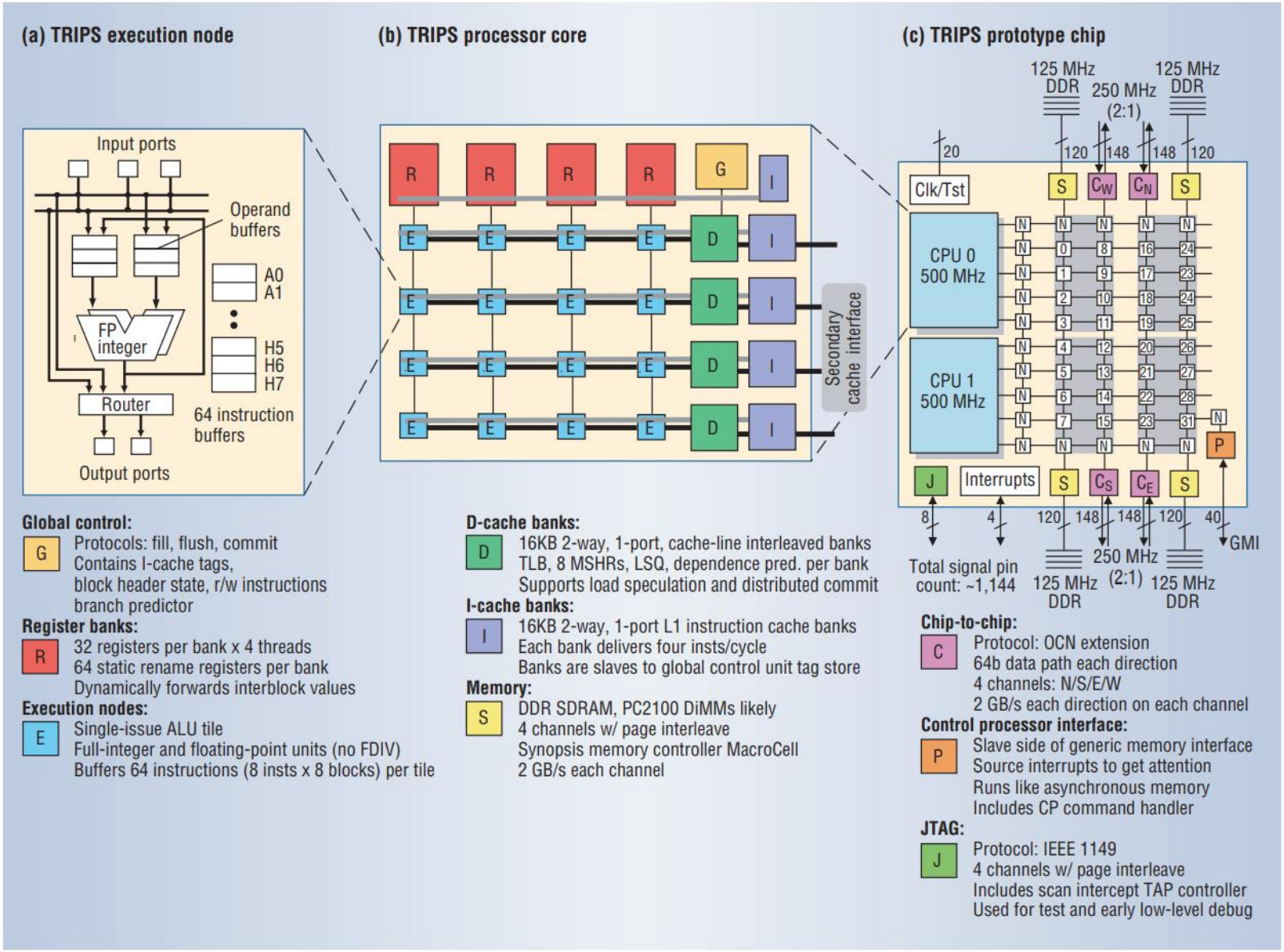


Figure 1. TRIPS prototype microarchitecture. (a) The prototype chip contains two processing cores, each of which is a 16-wide out-of-order issue processor that can support up to 1,024 instructions in flight. Each chip also contains 2 Mbytes of integrated L2 cache, organized as 32 banks connected with a lightweight routing network, as well as external interfaces. (b) The processor core is composed of 16 execution nodes connected by a lightweight network. The compiler builds 128-instruction blocks that are organized into groups of eight instructions per execution node. (c) Each execution node contains a fully functional ALU, 64 instruction buffers, and a router connecting to the lightweight inter-ALU network.

https://www.cs.utexas.edu/users/cart/trips/publications/computer04.pdf

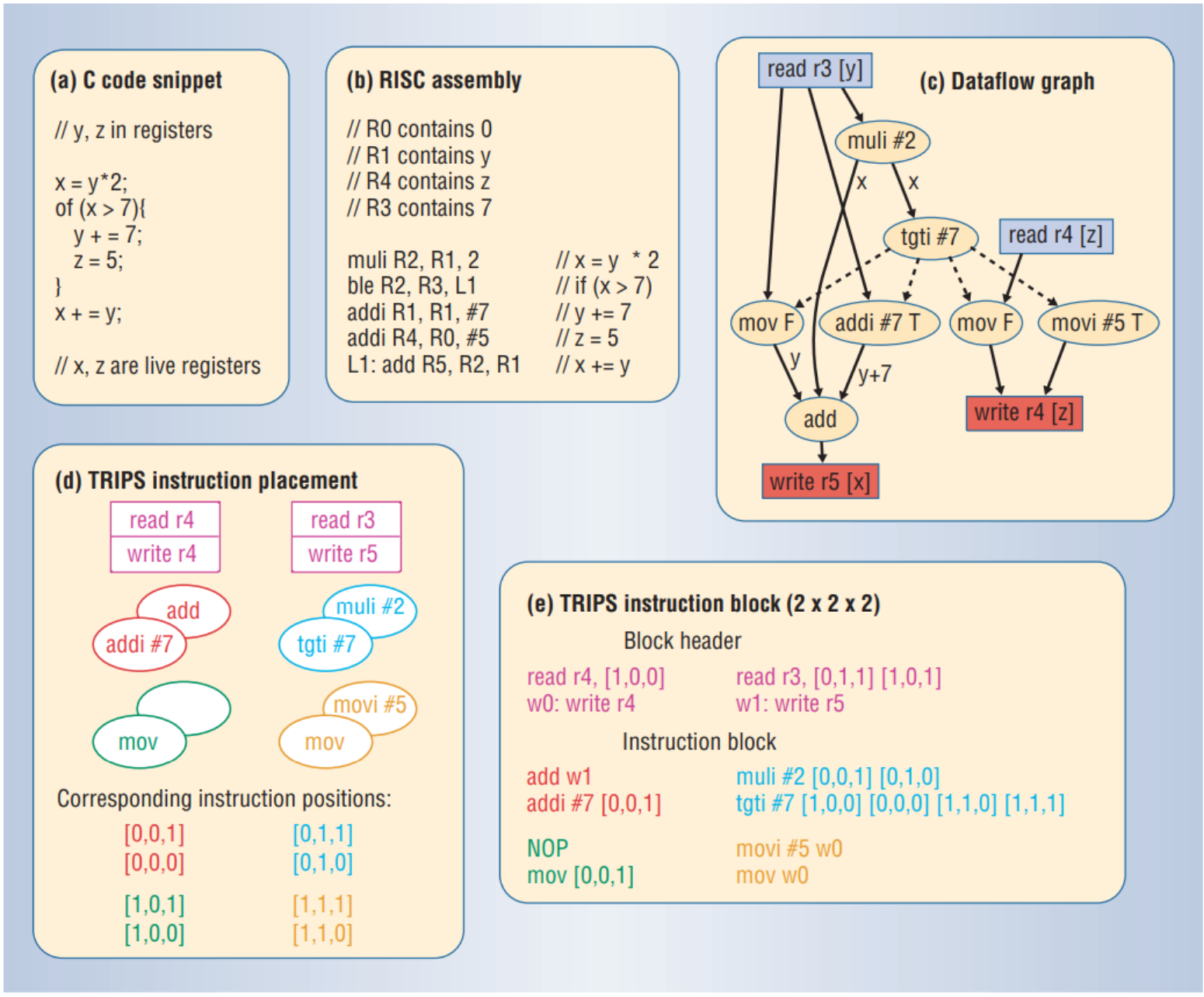
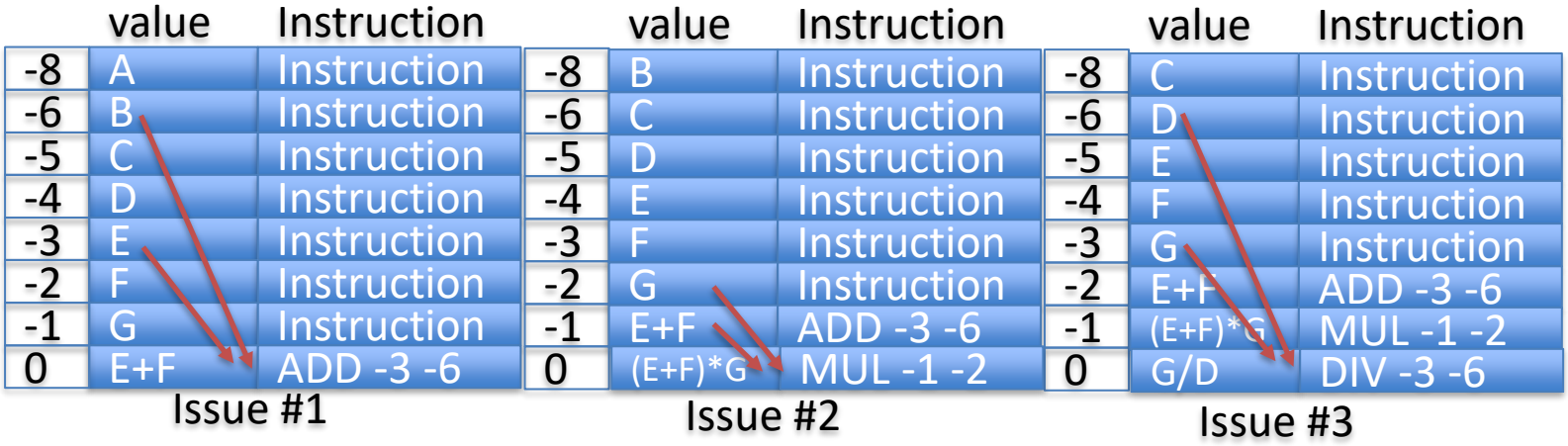


Figure 2. TRIPS code example. (a) In the C code snippet, the compiler allocates the input integer y to a register. (b) In the MIPS-like assembly code, the variable x is saved in register 5. (c) The compiler converts the original branch to a test instruction and uses the result to predicate the control-dependent instructions, which appear as dotted lines in the dataflow graph. (d) Each node in the 2 × 2 execution node array holds up to two buffered instructions. (e) The compiler generates in target form, which correspond to the map of instruction locations at the bottom of part (d).

https://www.cs.utexas.edu/users/cart/trips/publications/computer04.pdf

“Belt” (see also Fujitsu STRAIGHT)

- Idea:
  - every instruction writes to a fresh register
  - We have a finite rolling window of registers
  - Instructions collect operands using an offset backward to the instruction that produces what it needs



- At each instruction, the register window shifts
- Note that the register window now looks like the ROB

- What happens if we need A at issue #2 – it’s fallen out of the window!
- What happens with if-then – when control flow rejoins?

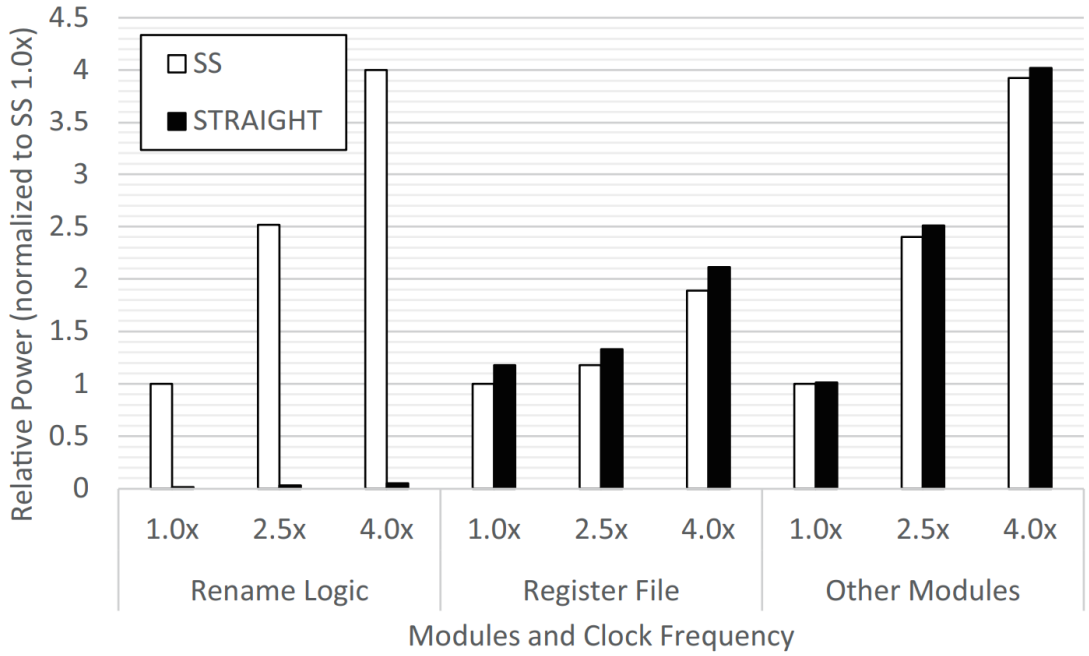


Fig. 17. Relative Power of SS and STRAIGHT with various clock frequency

- The claim is that this idea massively simplifies the front-end of the processor, reducing energy and misprediction penalty

- Hidetsugu Irie et al, *STRAIGHT: hazardless processor architecture without register renaming*. In Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-51). DOI:https://doi.org/10.1109/MICRO.2018.00019

References:

- Sinha, Steve & Chatterjee, Satrajit & Ravindran, Kaushik. (2019). **BOOST: Berkeley's Out-of-Order Stack Thingy.** https://www.researchgate.net/publication/228556746_BOOST_Berkeley's_Out-of-Order_Stack_Thingy
- Hidetsugu Irie et al, **STRAIGHT: hazardless processor architecture without register renaming**. In Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-51). DOI:https://doi.org/10.1109/MICRO.2018.00019
- Mill Computing: https://millcomputing.com/docs/
- Schmit, Herman; Benjamin Levine; Benjamin Ylvisaker (2002). **Queue Machines: Hardware Compilation in Hardware**. *10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'02)*: 152. doi:10.1109/FPGA.2002.1106670
- Aaron Smith et al, *Compiling for EDGE Architectures*. In Proceedings of the International Symposium on Code Generation and Optimization (CGO '06). DOI:https://doi.org/10.1109/CGO.2006.10
- Steven Swanson et al, **Area-Performance Trade-offs in Tiled Dataflow Architectures**. In Proceedings of the 33rd annual international symposium on Computer Architecture(ISCA '06). IEEE Computer Society, Washington, DC, USA, 314-326. DOI: https://doi.org/10.1109/ISCA.2006.10
- INTEL'S EXASCALE DATAFLOW ENGINE DROPS X86 AND VON NEUMANN. Aug 2018, Timothy Prickett Morgan, https://www.nextplatform.com/2018/08/30/intels-exascale-dataflow-engine-drops-x86-and-von-neuman/ . See also https://en.wikichip.org/wiki/intel/configurable_spatial_accelerator and US Patent No. US20180189231A1.