Lecture 9
Fundamental Processor Concepts

Xuan ‘Silvia’ Zhang
Washington University in St. Louis

http://classes.engineering.wustl.edu/ese566/

Instruction Set Architecture (ISA)

- Contract between software and hardware
  - representations for characters, integers, floating-point
  - integer formats can be signed or unsigned
  - floating-point formats can be single or double precision
  - byte address can be ordered within a word as either little or big-endian

  - Registers: general-purpose, floating-point, control
  - Memory: different addresses for heap, stack, I/O

Instruction Set Architecture (ISA)

- ISA Type
  - Register: operand stored in registers
  - Immediate: operand is an immediate in the instruction
  - Direct: address of operand in memory is stored in instruction
  - Register Indirect: address of operand in memory is stored in register
  - Displacement: register indirect, addr is added to immediate
  - Autoincrement/decrement: register indirect, addr is automatically adjusted
  - PC-Relative: displacement is added to the program counter

Instruction Set Architecture (ISA)

- ISA Category
  - integer and floating-point arithmetic instructions
  - register and memory data movement instructions
  - control transfer instructions
  - system control instructions

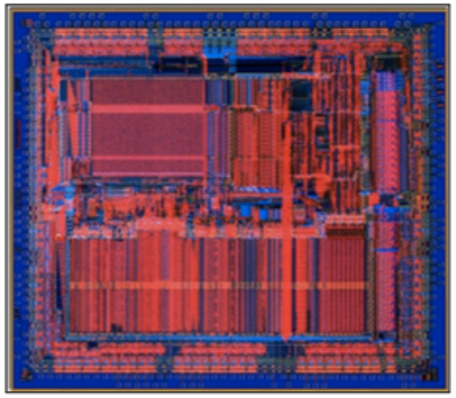
- ISA Style
  - opcode, addresses of operands and destination, next instruction
  - variable length vs. fixed length

MIPS32 ISA

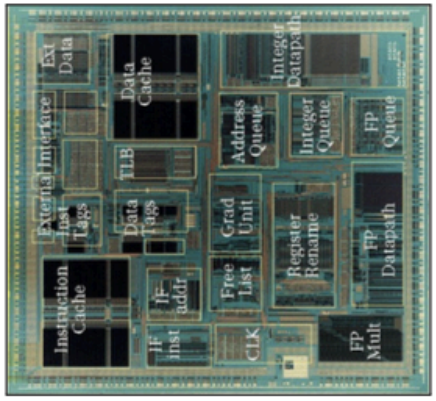
- how is data represented?
  - 8-bit bytes, 16-bit half-words, 32-bit words
  - 32-bit single-precision, 64-bit double-precision floating point
- where can data be stored?
  - 232 32-bit memory locations
  - 32 general-purpose 32-bit registers, 32 SP (16 DP) floating-point registers
  - FP status register, program counter

MIPS32 ISA

- how can data be accessed?
  - register, register indirect, displacement
- what operations can be done on data?
  - large number of arithmetic, data movement, and control instructions
- how are instructions encoded?
  - fixed-length 32-bit instructions

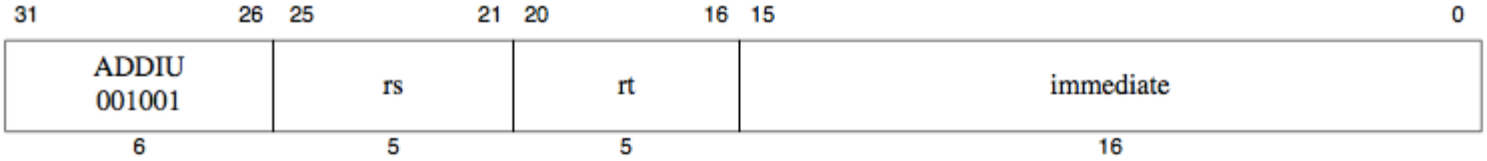


MIPS R2K: 1986, single-issue, in-order, off-chip caches, 2 µm, 8–15 MHz, 110K transistors, 80 mm^2



MIPS R10K: 1996, quad-issue, out-of-order, on-chip caches, 0.35 µm, 200 MHz, 6.8M transistors, 300 mm^2

MIPS32 Instruction Example



Format: `ADDIU rt, rs, immediate` **MIPS32**

Purpose: Add Immediate Unsigned Word

To add a constant to a 32-bit integer

Description: `GPR[rt] ← GPR[rs] + immediate`

The 16-bit signed *immediate* is added to the 32-bit value in GPR *rs* and the 32-bit arithmetic result is placed into GPR *rt*.

No Integer Overflow exception occurs under any circumstances.

Restrictions:

None

Operation:

```
temp ← GPR[rs] + sign_extend(immediate)
GPR[rt] ← temp
```

Exceptions:

None

Programming Notes:

The term "unsigned" in the instruction name is a misnomer; this operation is 32-bit modulo arithmetic that does not trap on overflow. This instruction is appropriate for unsigned arithmetic, such as address arithmetic, or integer arithmetic environments that ignore overflow, such as C language arithmetic.

PARC ISA

- Subset of MIPS32 with important differences
  - only little-endian, very simple address translation
  - no hi/lo registers, only 32 general purpose registers
  - multiply and divide instructions target general purpose registers
  - only a subset of all MIPS32 instructions
  - no branch delay slot

- PARCv1
  - very small subset suitable for examples
  - addu, addiu, mul
  - lw, sw
  - j, jal, jr
  - bne
  - mfc0, mtc0 (proc2mngr, mngr2proc), nop

PARC ISA

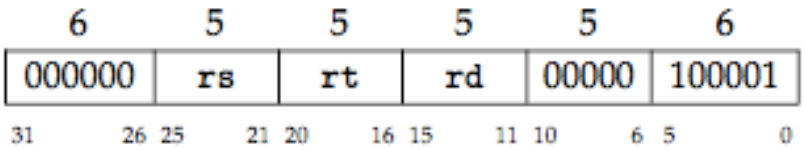
- PARCv2
  - subset suitable for executing simple C programs without system calls (i.e., open, write, read)
  - subu, and, or, nor, xor, andi, ori, xori, lui
  - slt, sltu, slti, sltiu, sll, srl, sra, srav, srlv, sllv
  - bgtz, bltz, bgez, blez
  - mfc0, mtc0 (stats_en, core_id, num_cores)
- PARCv3
  - single-threaded and parallel programs with system calls
  - jalr
  - lb, lbu, lh, lhu, sb, sh
  - amo.add, amo.and, amo.or, sync
  - syscall, eret
  - add.s, sub.s, mul.s, div.s, c.<cond>.s, cvt.s.w, trunc.w.s
  - div, divu, rem, remu
  - movn, movz
  - mtx, mfx, mtxr, mfxr

PARCv1 Assembly, Semantics, and Encoding

`addu rd, rs, rt`

R[rd] ← R[rs] + R[rt]

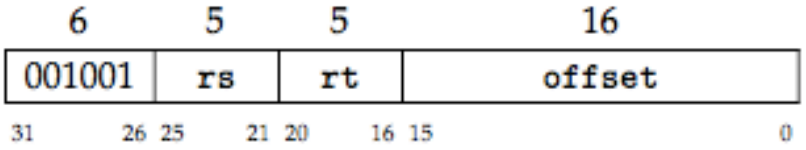
PC ← PC+4



`addiu rt, rs, imm`

R[rt] ← R[rs] + sext(imm)

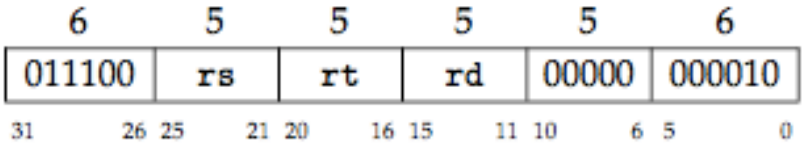
PC ← PC+4



`mul rd, rs, rt`

R[rd] ← R[rs] × R[rt]

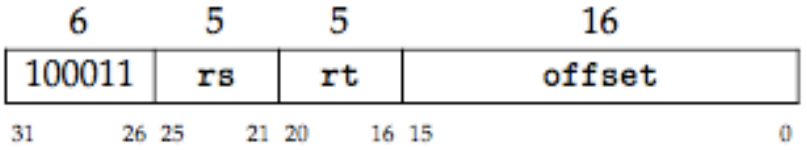
PC ← PC+4



`lw rt, offset(rs)`

R[rt] ← M[R[rs] + sext(offset)]

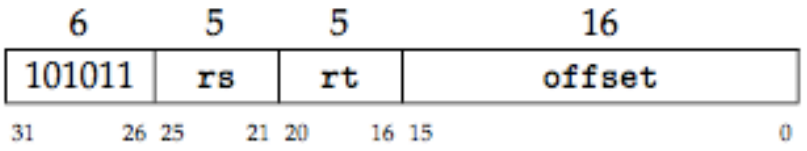
PC ← PC+4



`sw rt, offset(rs)`

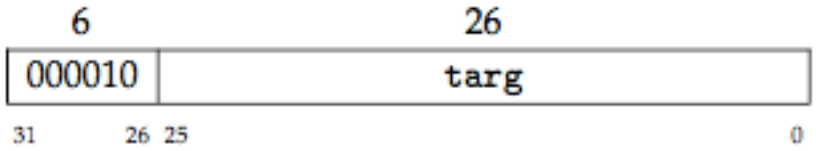
M[R[rs] + sext(offset)] ← R[rt]

PC ← PC+4

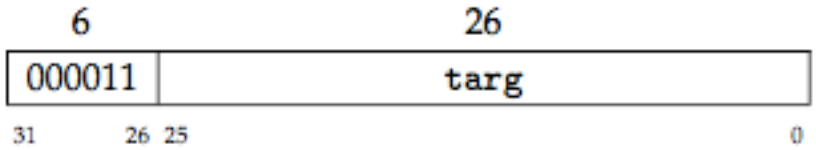


PARCv1 Assembly, Semantics, and Encoding

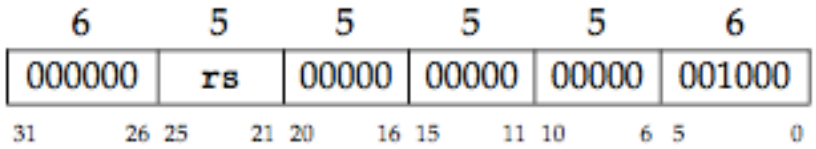
```
j targ
```
PC ← { (PC + 4)[31:28], targ, 00 }



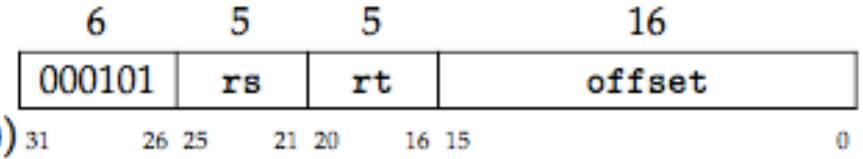

```
jal targ
```
R[31] ← PC + 4;
PC ← { (PC + 4)[31:28], targ, 00 }



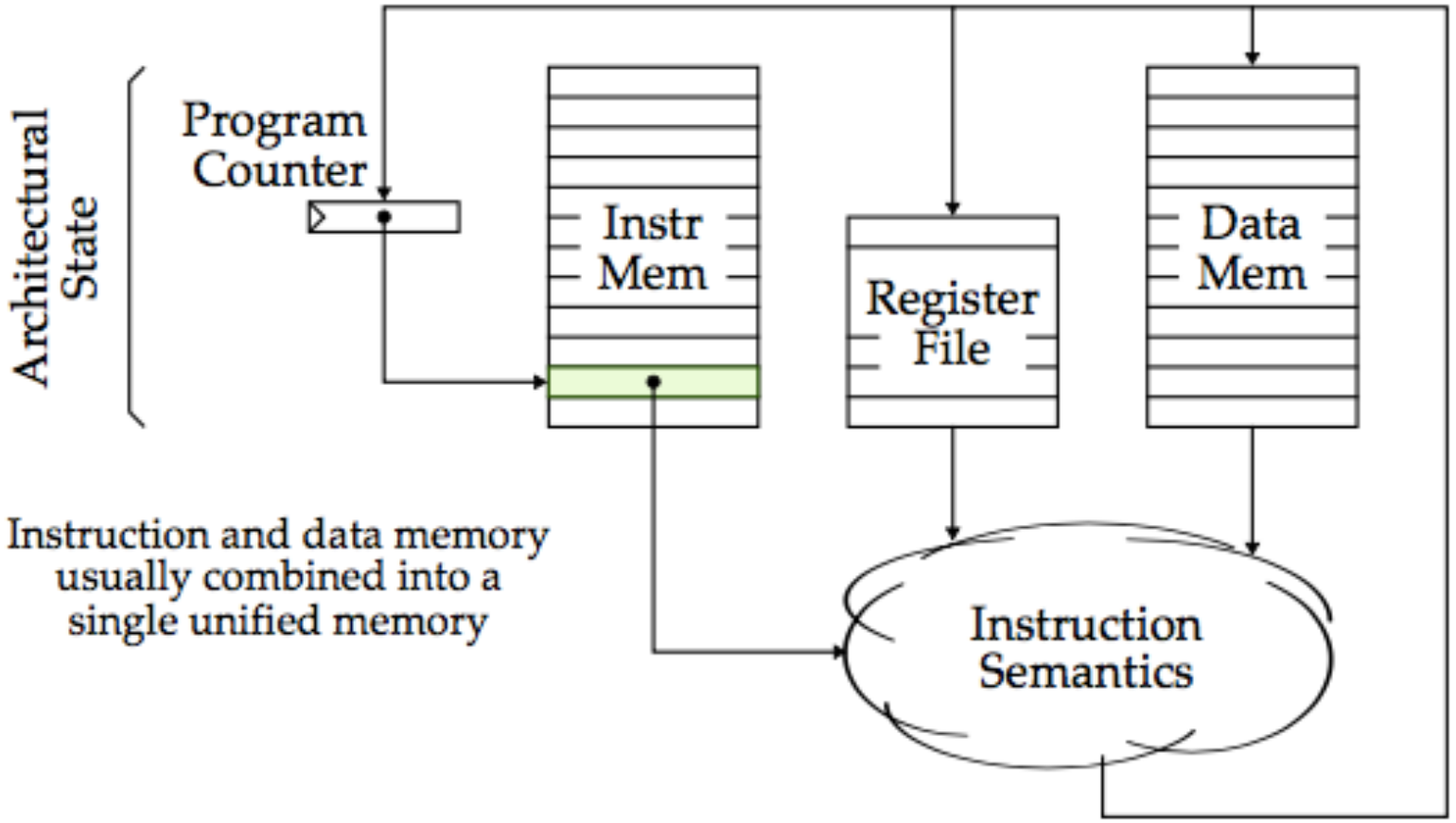

```
jr rs
```
PC ← R[rs]




```
bne rs, rt, offset
if ( R[rs] != R[rt] )
 PC ← ( PC + 4 + (4 × sext(offset))
```



Processor Functional-Level Model



Transactions and Steps

- Each instruction as a transaction
- Executing a transaction involves a sequence of steps

	addu	addiu	mul	lw	sw	j	jal	jr	bne
Fetch Instruction									
Decode Instruction									
Read Registers									
Register Arithmetic									
Read Memory									
Write Memory									
Write Registers									
Update PC									

Transactions and Steps

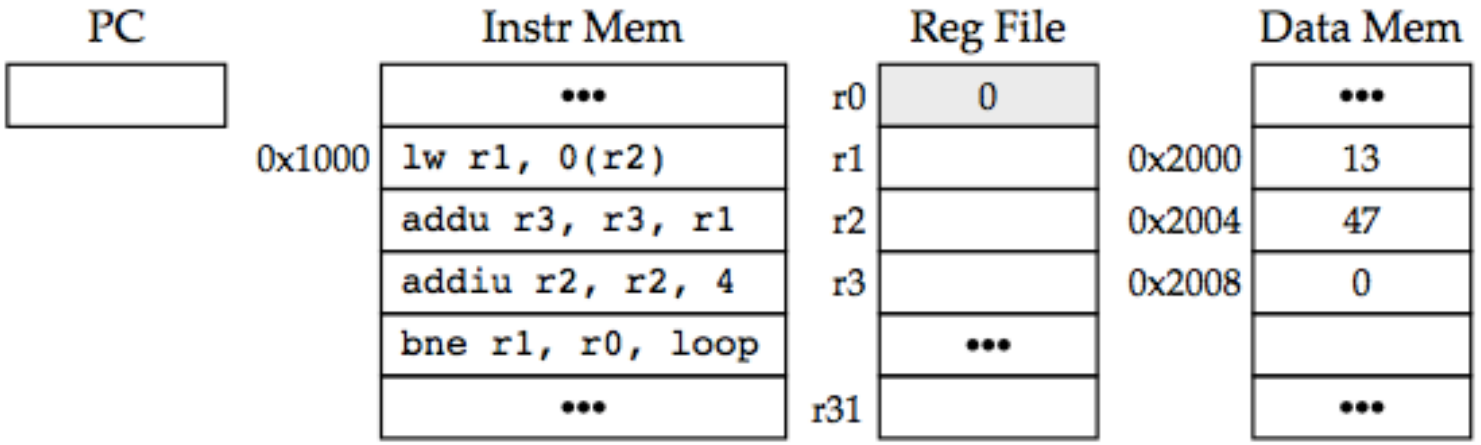
- Each instruction as a transaction
- Executing a transaction involves a sequence of steps

	addu	addiu	mul	lw	sw	j	jal	jr	bne
Fetch Instruction	✓	✓	✓	✓	✓	✓	✓	✓	✓
Decode Instruction	✓	✓	✓	✓	✓	✓	✓	✓	✓
Read Registers	✓	✓	✓	✓	✓			✓	✓
Register Arithmetic	✓	✓	✓	✓	✓				✓
Read Memory				✓					
Write Memory					✓				
Write Registers	✓	✓	✓	✓			✓		
Update PC	✓	✓	✓	✓	✓	✓	✓	✓	✓

Simple Assembly Example

Static Asm Sequence	Instruction Semantics
`loop: lw r1, 0(r2)`	
`addu r3, r3, r1`	
`addiu r2, r2, 4`	
`bne r1, r0, loop`	

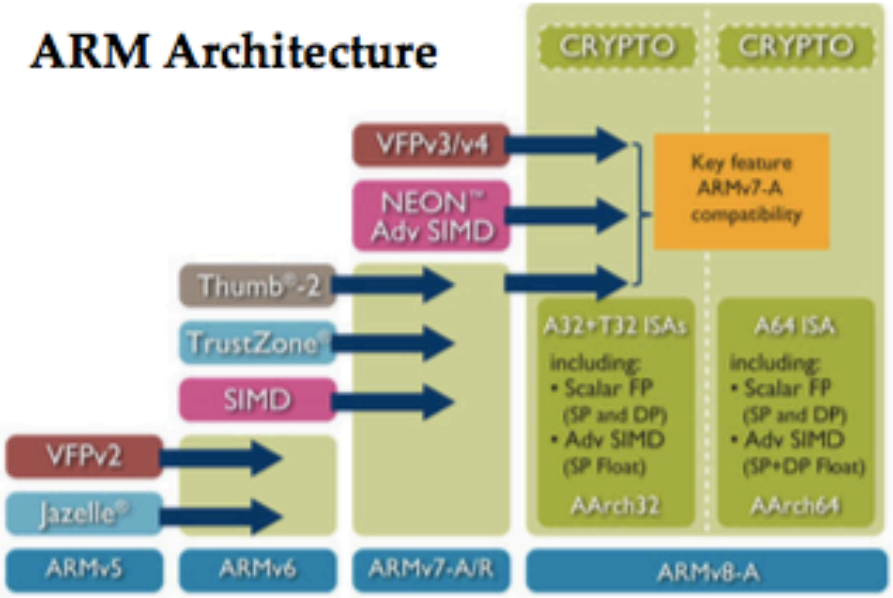
Worksheet illustrating processor functional-level model

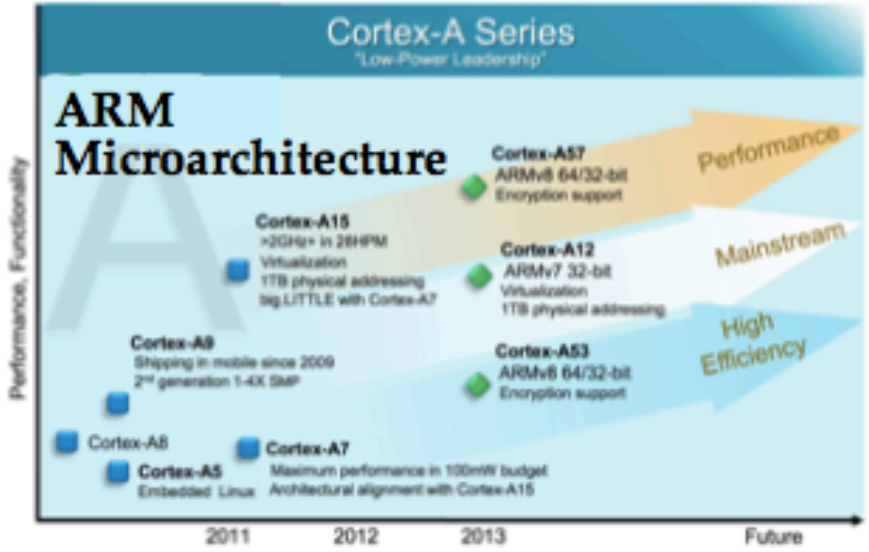


Processor/Laundry Analogy

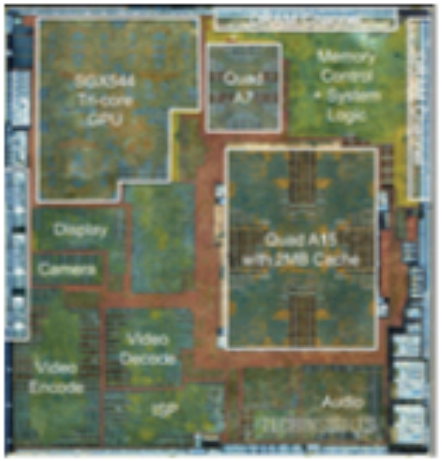
- Processor
  - instructions are “transactions” that execute on a processor
  - architecture: defines the hardware/software interface
  - microarchitecture: how hardware executes sequence of instructions
- Laundry
  - cleaning a load of laundry is a “transaction”
  - architecture: high-level specification, dirty clothes in, clean clothes out
  - microarchitecture: how laundry room actually processes multiple loads

Arch vs. µArch vs. VLSI Implementation

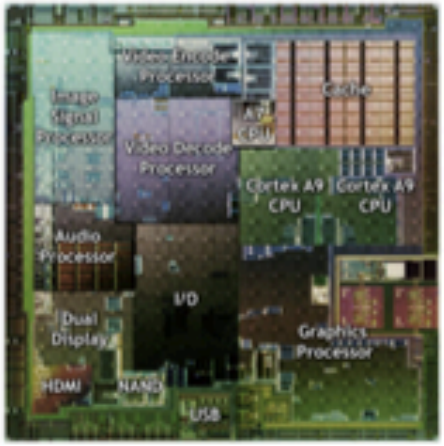




ARM VLSI Implementation

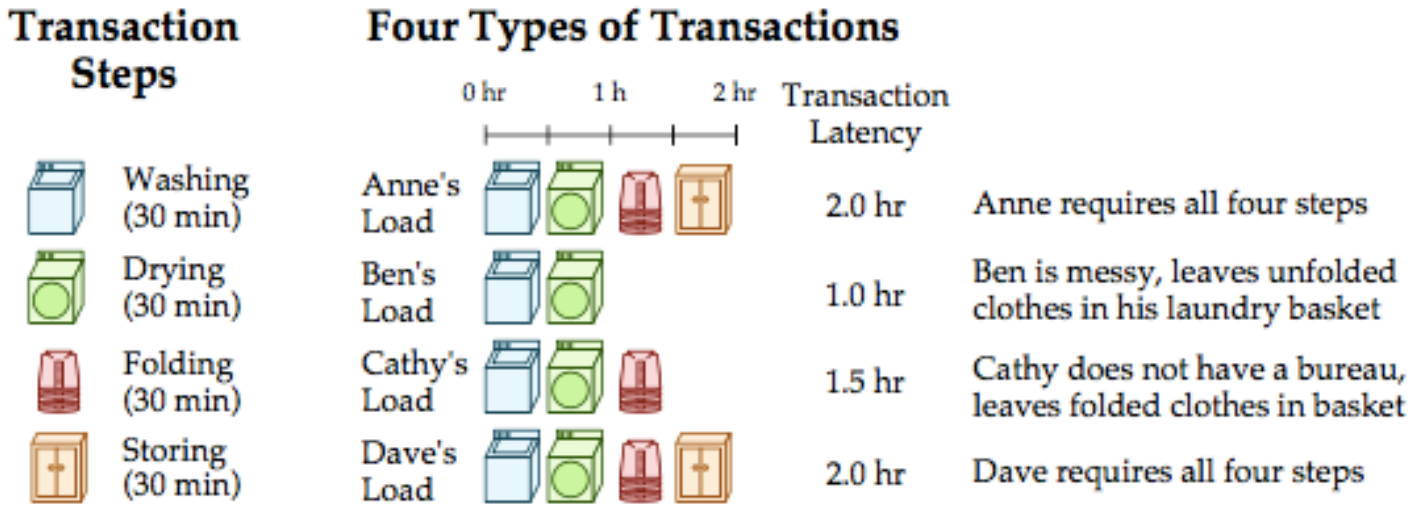


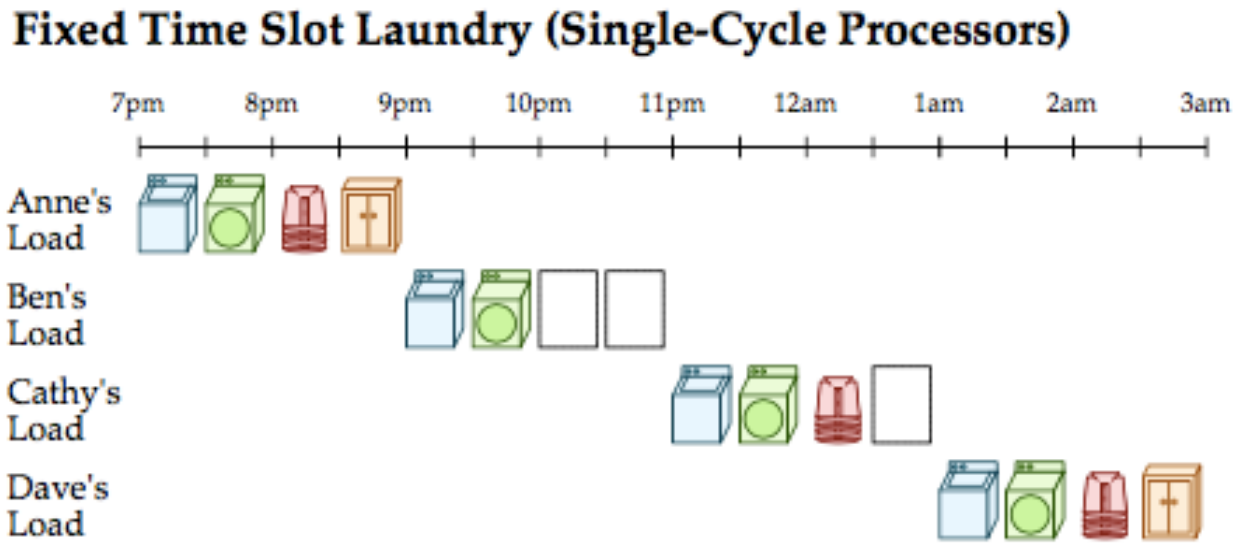
Samsung Exynos Octa



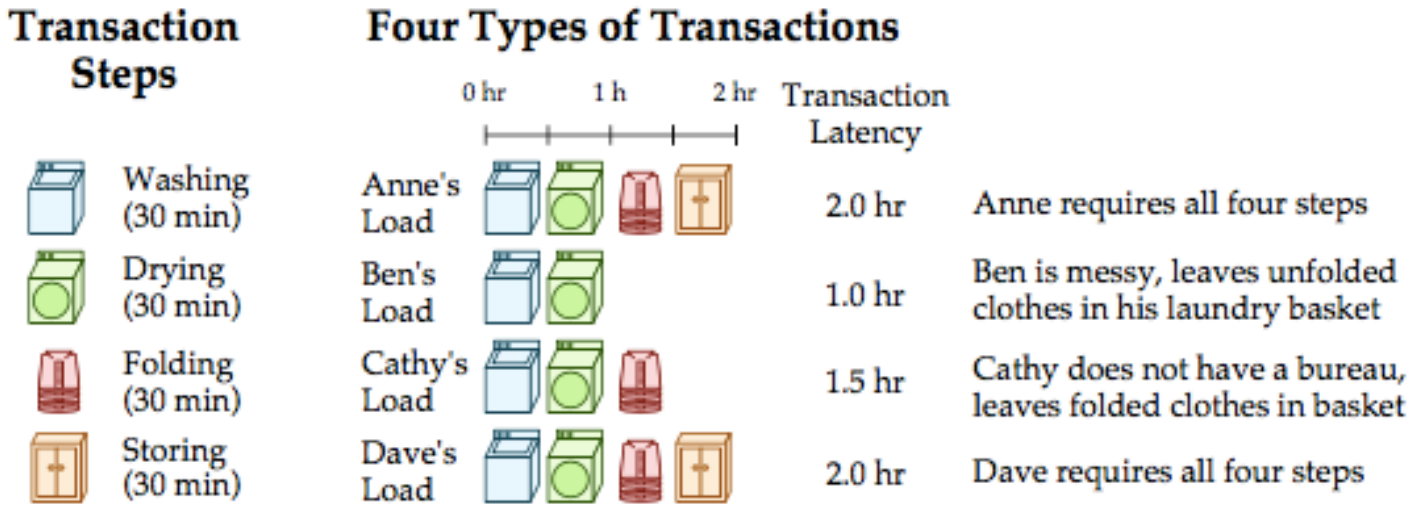
NVIDIA Tegra 2

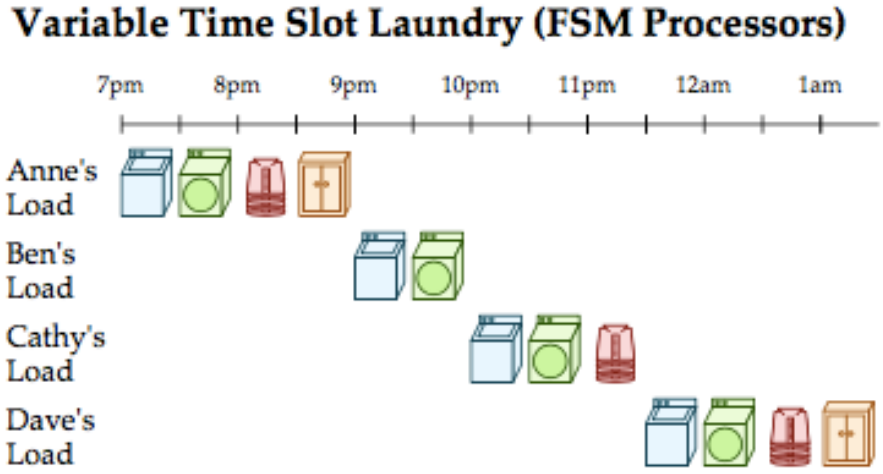
Processor Microarchitecture Design Patterns

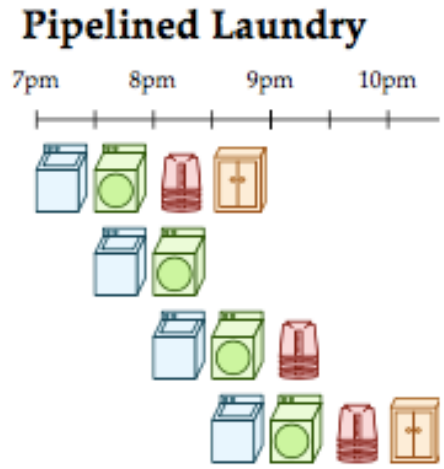




Processor Microarchitecture Design Patterns







Lab2: Explore Integer Multiplier Designs

- Single-cycle design
  - fixed-latency iterative multiplier
  - one-cycle multiplier
- FSM
- Pipelined
- Design trade-offs
  - how does performance, area, and power relate
- Due on 2/22 at 2:30pm

Questions?

Comments?

Discussion?

Acknowledgement

Cornell University, ECE 4750