Processing Elements

Road map:
- models of computation (system spec.)
  → KPN, SDF, StateCharts
- performance estimation (system design and validation)
  → SystemC
- system architecture (system design)
  → PEs, Memory, On-chip inter-connect

PE taxonomy

1. IP instruction processors
   - execute program according to the ISA
2. HW accelerators
   - execute one (or more) fixed functions
3. HW Coprocessors
   - extends ISA

Instruction Processors Types

1. General Purpose
   - designed for a variety of uses
   - measured against benchmarks
     - e.g. SPEC CPU2017 (sequential) www.spec.org
     - e.g. PARSEC (parallel) cs.princeton.edu
     - e.g. EEMBC (embedded) www.eembc.org

2. Application-Specific
   - DSP: digital signal processors
   - GPU: image processing / numerical computing

Instruction Processor Performance

- improve performance by increasing:
  - clock rate
  - instruction parallelism

1. increasing clock rate:

   option i) increase voltage
   - clock frequency is roughly linear w.r.t. voltage
   - dynamic (switching) power: $P = CV^2f$
     C ← of transistors & wires

   $\Rightarrow$ switching power $\propto f^3$
   - challenges: distributing power, removing heat

option ii) deeper pipeline
- deeper pipeline ⇒ shorter stages ⇒ faster clock without increasing voltage
- need good branch prediction and out-of-order execution ⇒ consume power
- pipeline depths

			(2010)	(2019)
ARM 7 (1993)	ARM 11 (2002)	Cortex A9 (2007)	Cortex A15	Cortex A77
3	8	8	15	13
Pentium (1993)	Pentium 3 (1999)	Pentium 4	Core (2006)	Atom (2008)
5	10	20-31	14-19	12-14

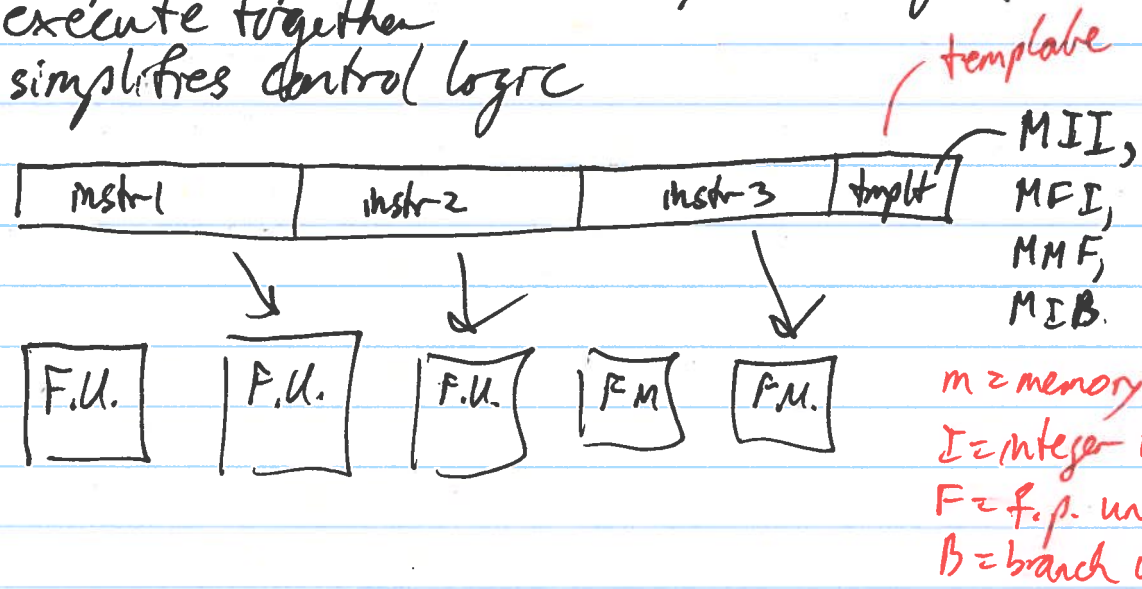
hit the "power wall"

2) increasing instruction parallelism

option i) superscalar datapath
- fetch, decode, dispatch, execute, commit 2/4/6/8 instr. per cycle
- extracts instruction-level parallelism (ILP) from a thread of execution (transparent to the software)
- decode widths:

ARM 11	Cortex A9	Cortex A15	Cortex A77
1 (scalar)	2	3	4
Pentium	Core	Apple M1	IBM Power
2	4	8	8

option ii) VLIW
- very long instruction word
- compiler specifies ILP by bundling operations to execute together
- simplifies control logic



- VLIW processors:
  DSPs (TMS320)
  Intanium (Intel IA-64 arch.)
  Nvidia Bluefield-2 accelators
  Qualcomm Hexagon DSP

option iii) parallel programing
- simultaneous multi-threading (thread-level parallelism)
  - single core has state for 2+ threads (registers, PC, status register)
  - the threads share the functional units
  - better utilization of resources
  - not popular in embeded systems - thread performance depends on other threads on the core
- multiprocessing (program-level parallelism)
  - use multiple PEs
  - memory / interconnect can be a bottleneck

DSP Digital Signal Processors

- processors optimised for sequential math operations on streaming data
  - e.g. TI TMS320 (1983–present)
    - fixed-point and floating-point variants
  - e.g. Qualcomm Hexagon (2009–present)
    - VLIW
    - was fixed-point, now floating-point
    - in Qualcomm Snapdragon SoC (CPU, DSP, GPU, GPS, cellular radio, NFC, image processor)
- features:
  - multiply-accumulate:
    - MACa R_i, R_j ; $R_a \leftarrow R_a + R_i * R_j$
  - circular address registers
    - increment modulo defined buffer size
  - single loop instruction
    - RPT i – loops next instruction i times

```
float *inputs = SOME_BUFFER, coeff[LARGE_VALUE] = CONSTANTS,
        res = 0;
int ii = START_VALUE;   initial index into circular input buffer
for(int c=0; c< LARGE_VALUE; c++)
    res += input[(ii+c)% LARGE_VALUE] * coeff[c];
```

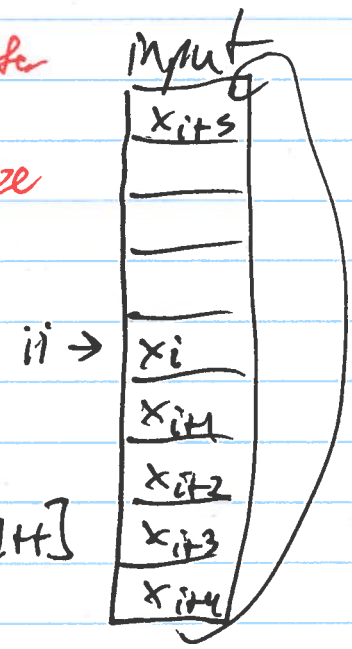
circular

index to input { MOV Rca0, START_VALUE circular buffer index
MOV Rcs0, LARGE_VALUE circ. buff. size

coeff { MOV Rca1, 0
MOV Rcs1, LARGE_VALUE

accum { MOV Ra, 0

```
RPT LARGE_VALUE
    MACa  inputs[Rca0++], coeff[Rca1++]
STR res, Ra
```

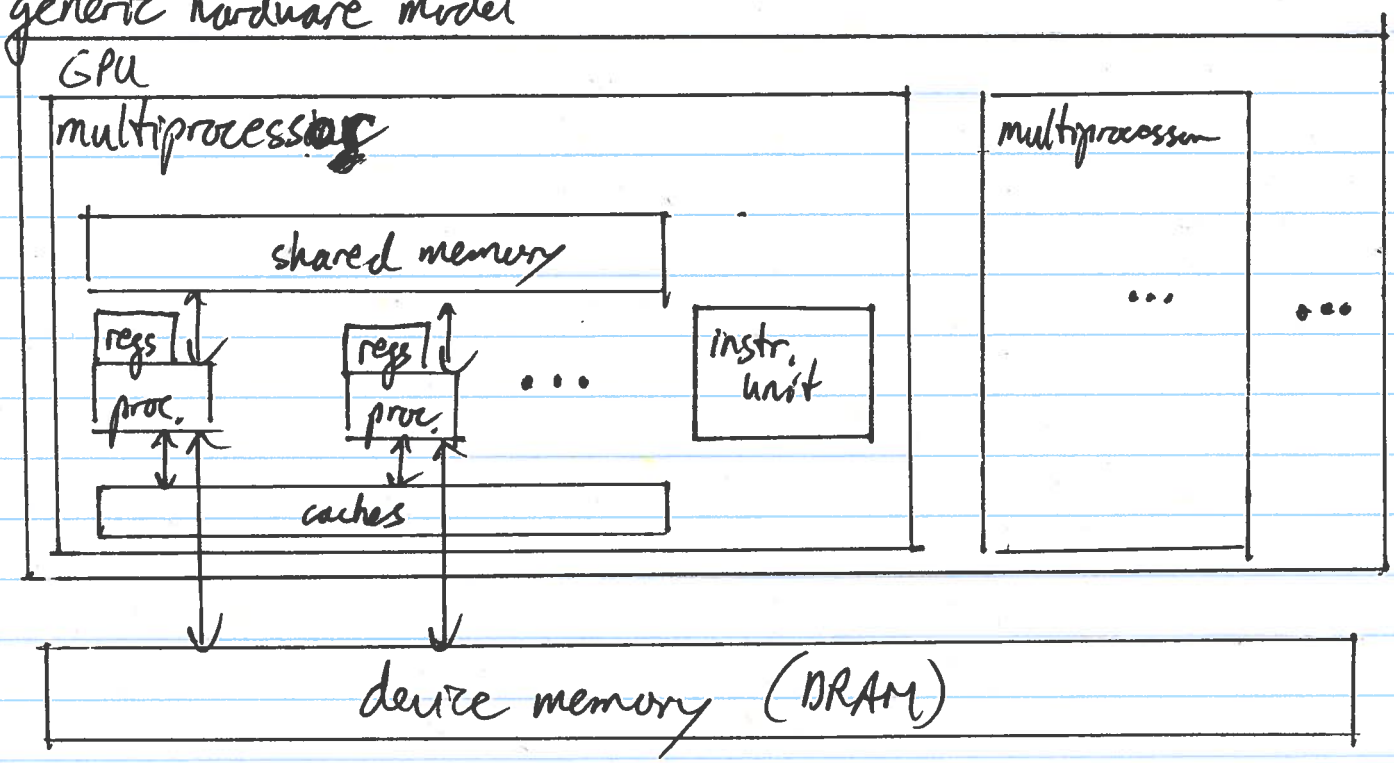


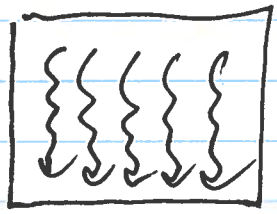
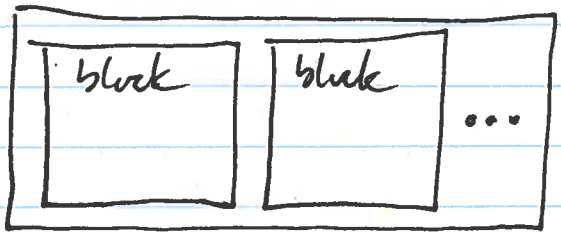
! instr. loops
vs multiple instructions on a general-purpose processor

GPGPU: General Purpose GPU Programming

top500.org

- GPUs can do thousands of floating-point operations in parallel
- gpgpu frameworks:
  - CUDA - proprietary (Nvidia)
  - OpenCL - open (AMD, IBM, Intel, Nvidia, Qualcomm, ...)
- generic hardware model



- programming model
  - threads 
    - executes an instance of a kernel
    - unique thread id
  - blocks 
    - set of threads executing the same code
    - unique block ids
    - blocks can be multi-dimensional (x,y,z)
  - grid 

    - array of blocks running same kernel
    - can be multi-dimensional (x,y,z)
- execution model
  - blocks get assigned to multiprocessors
  - threads are executed in warps of (32, 64) executing the instructions in lockstep
  - they can use predication for conditional execution

e.g.

```
var = 0;                    mov.s32   var, 0
if(condition)               setp      p, condition
    var = 1;            @p  mov.s32   var, 1
else                    @!p mov.s32   var, 2
    var = 2;
```

handouts > cuda Saxpy

CUDA SAXPY Example

Source: https://devblogs.nvidia.com/easy-introduction-cuda-c-and-c/

SAXPY: Single-precision A * X Plus Y

```
#include <stdio.h>

__global__
void saxpy(int n, float a, float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int main(void)
{
  int N = 1<<20;
  float *x, *y, *d_x, *d_y;
  x = (float*)malloc(N*sizeof(float));
  y = (float*)malloc(N*sizeof(float));

  cudaMalloc(&d_x, N*sizeof(float));
  cudaMalloc(&d_y, N*sizeof(float));

  for (int i = 0; i < N; i++) {
    x[i] = 1.0f;
    y[i] = 2.0f;
  }

  cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

  // Perform SAXPY on 1M elements
  saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);
  cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);

  float maxError = 0.0f;
  for (int i = 0; i < N; i++)
    maxError = max(maxError, abs(y[i]-4.0f));
  printf("Max error: %f\n", maxError);

  cudaFree(d_x);
  cudaFree(d_y);
  free(x);
  free(y);
}
```

~~max~~ array size

coefficient

"kernel"

1M

in ~~pro~~ CPU's memory

device memory

initialize in CPU memory

copy from CPU to GPU (DMA)

#blocks = ⌈N/block size⌉

block size

copy result from GPU to CPU

free device mem.

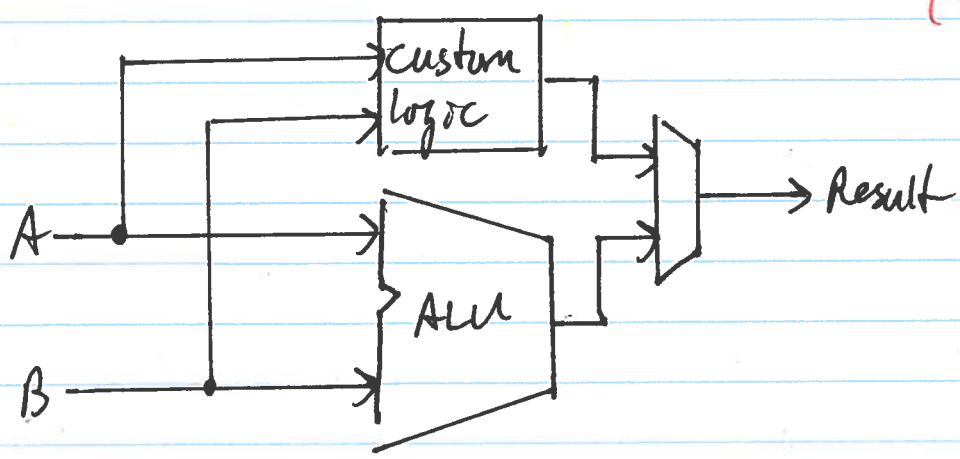
free cpu memory

ASIP: Application-Specific Instruction-set Processor

Types:
- configurable architecture
  - e.g. vary #/type of functional units
  - e.g. configure memory system (e.g. Nios II - MMU is optional, Caches are optional)
- ISA optimization (Instruction Set Architecture)
  - e.g. add custom instructions e.g. CRC instructional
  - e.g. remove unused ~~functions~~ instructions
- tools generate RTL and configure compilers

Extensible Processors
- Nios II (user-defined instructions)

(soft-core CPU)



- up to 256 custom instructions
- software macros are used to invoke it
  - e.g. int y = ALT_CI_CRC(x);

integer

- Xtensa (synthesized instructions)

e.g.

```
; use generator
; generator:: regfile(name => "L",
                      cname => "long128",
                      sname => "s",
                      width => 128);
opcode add128
add128 {assign sr = st + ss};
```

- specify a 16 × 128-bit reg. file

- given the above, the processor generator generates RTL, testbench, etc.
  - the RTL is automatically pipelined
  - also generates processor pipeline interlocks (hazard detection and stalling) and result bypass logic (data forwarding)

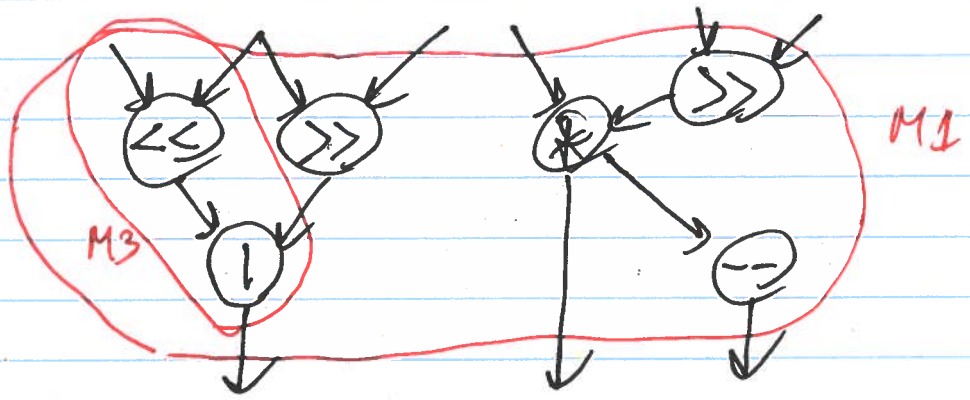
- to use the above in C:

```
int main() {
    long128 src1[N], src2[N], dst[N];
    for(int i=0; i<N; i++)
        dst[i] = add128(src1[i], src2[i]);
}
```

Instruction Set Synthesis

1. extract CFG for the application
2. analyze the DFG of each basic block and extract "cuts" (synthesized instructions)
3. estimate area and clock cycles for each cut
4. evaluate application speedup
5. pick the best set of cuts based on constraints (area)

e.g. basic block

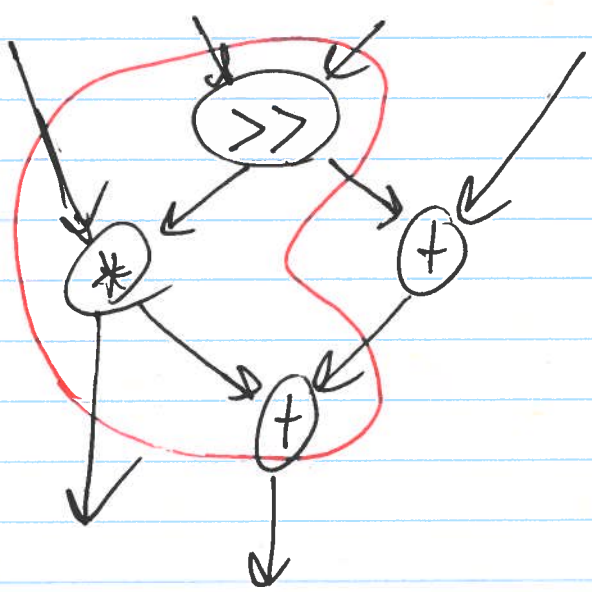


M3 : 3 inputs, 1 output

M1: 6 inputs, 3 outputs

might not be feasible based on reg file ports

e.g.



– illegal: can't be done in one instruction

- Speedup of a cut, C, can be computed as:

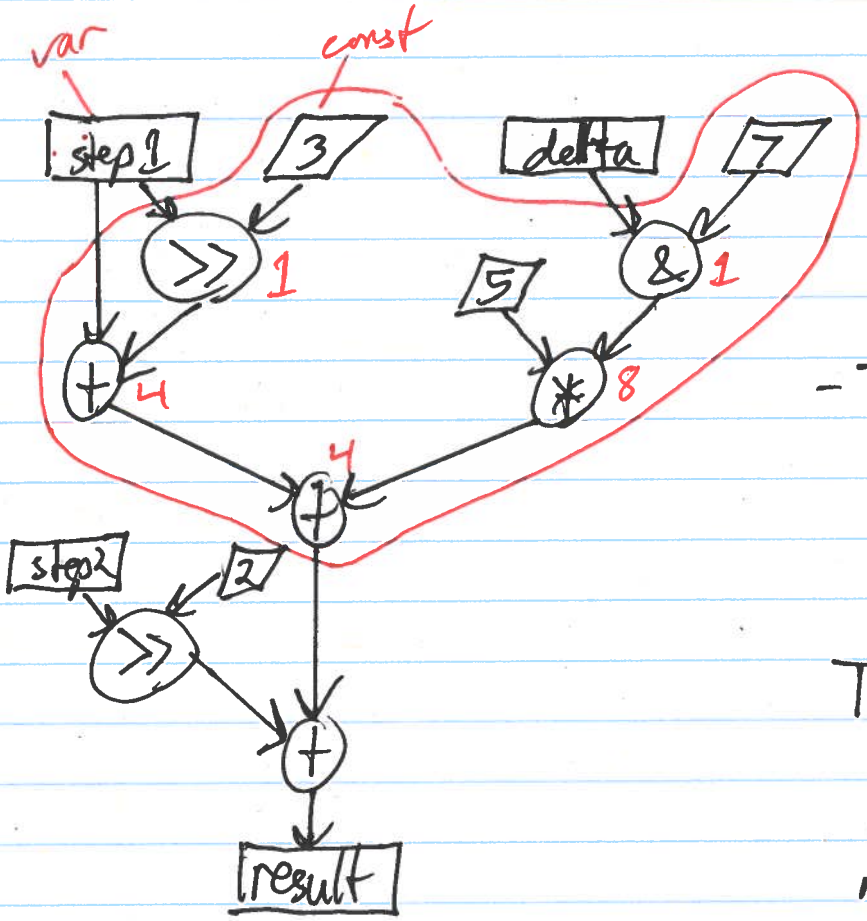
$$S = \frac{T_{tot}}{T_{tot} - n_c\left(T^c_{sw} - T^c_{hw}\right)}$$

original time

times cut is executed

difference in time of the cut

e.g. "PROC" block

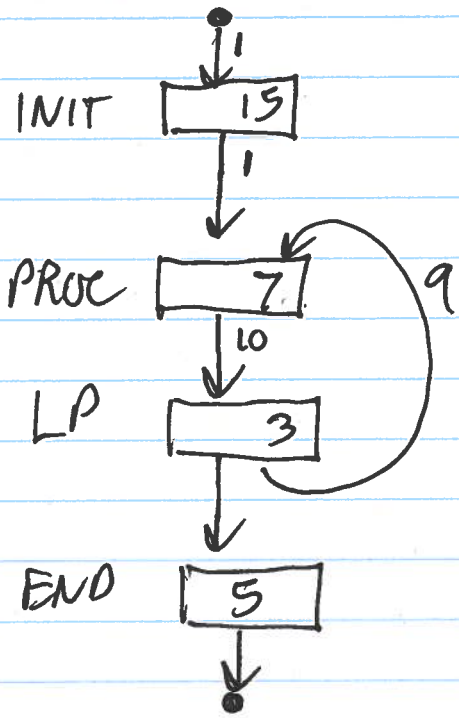


← largest cut with 2 inputs / 1 output?

- T^c_{sw}:
  - 5 operations/instructions
  - processor clock 100MHz
  - $5 \times 10ns = 50ns$

T^c_{hw}:
logic = 1 ns
adder = 4 ns
multiplier = 8 ns
- critical path = 13 ns
- $\lceil 13ns/10ns \rceil = 2$ cycles
- $2 \times 10ns = 20ns$

CFG (application)



$$T_{tot} = 15 + (7+3)*10 + 5$$
$$= 120 \text{ cycles}$$

$$n_c = 10$$
$$T^c_{sw} = 5, \quad T^c_{HW} = 2$$

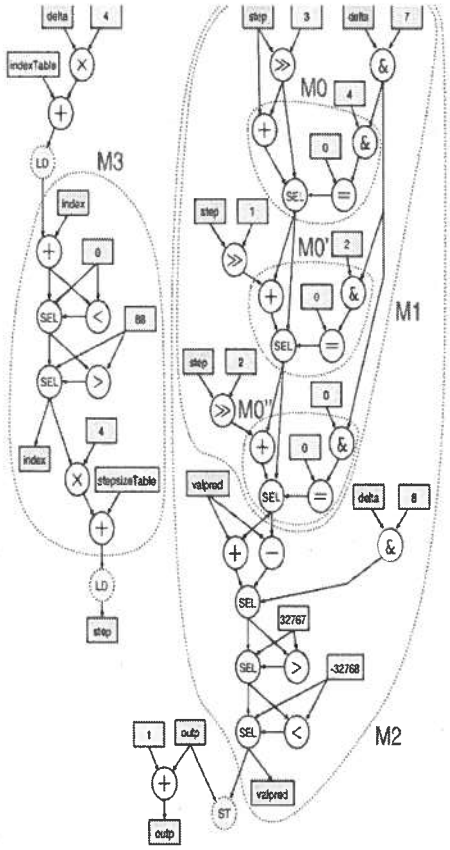
$$S = \frac{120 \text{ cycles}}{(120 - 10(5-2))} = \frac{120}{90} = 1.33$$

- choose a set of cuts that maximizes total speedup while staying within area constraints and register file limitations

Handouts > instr Synth

adpcm example

audio format



- SEL is a selector node for conditional execution
- can execute M3 and M2 in parallel
- M0 is executed multiple times but it is small and have a lot of inputs
- M1 is a 16x3-bit multiply with 2 inputs/1 output
- depending on hardware M1, M2, or M3 provide better speedup

Atasu, Pozzi, Ienne. *Automatic Application-Specific Instruction-Set Extensions under Microarchitectural Constraints*, Proceedings 40th Design Automation Conference, 2003.

Input/Output Multiplexing

- if #inputs/outputs > register file ports
- use mux/demux – takes multiple cycles

e.g. 4 inputs, 2 outputs (reg file 2 read ports, 1 write port)