Design Patterns for Embedded Software: Basics for embedded programming

課程：嵌入式系統與軟體工程

開發學校：台大電機系

王勝德 教授

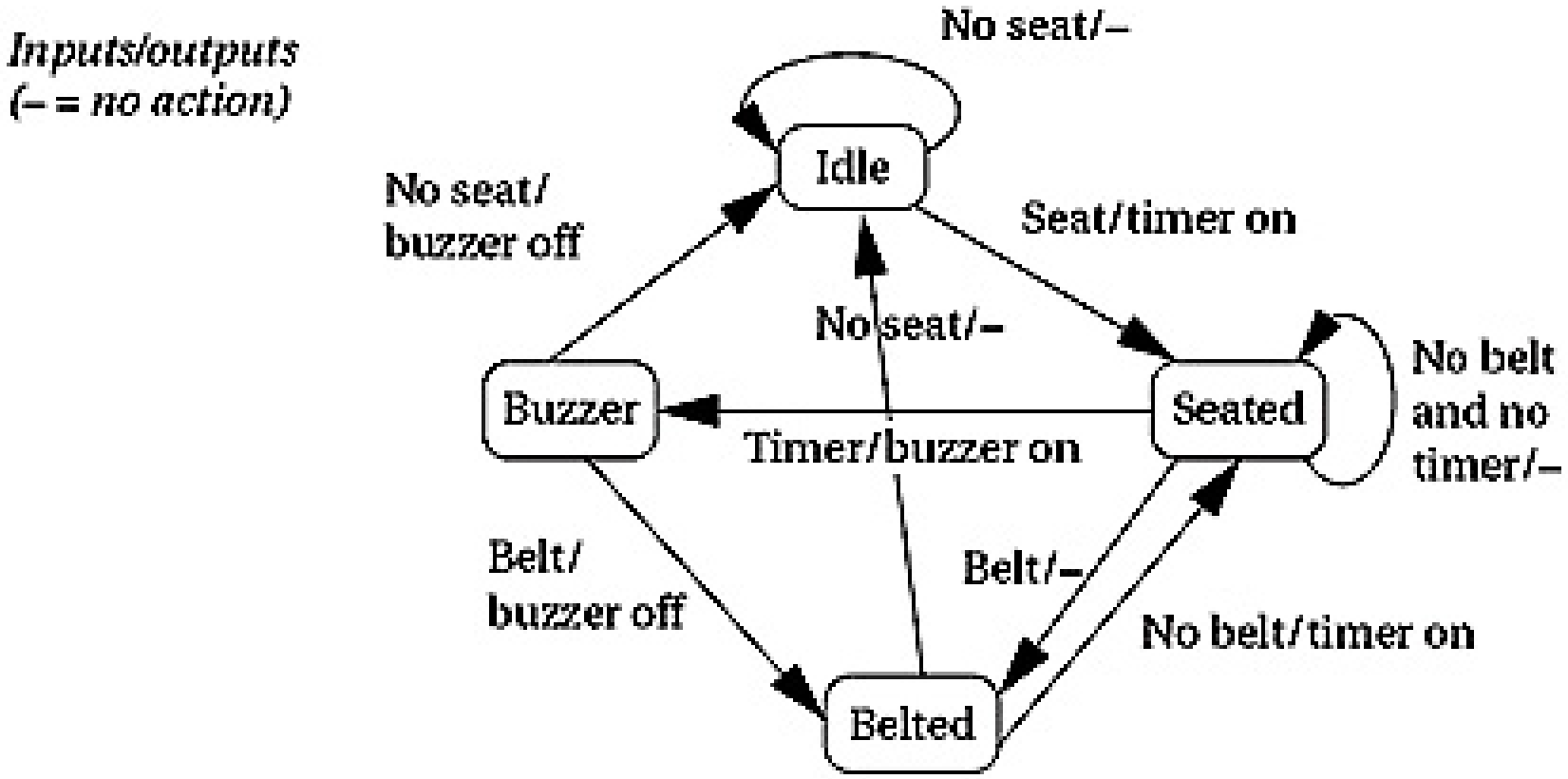
Outline

- Challenges of Embedded Systems Programming
- Design Patterns for Embedded Systems
- Models of Programs
- Analysis of Execution Time
- Performance Optimizations
- Power Saving Techniques
- Reducing Data Size
- Reducing Code Size
- Testing

Challenges of Embedded Systems Programming

- Run at a required rate to meet system deadline– time constraints
  - however, often with limited computing power
- Limited memory- space constraints
- Power consumption– energy constraints

Design Patterns – State Machine (1/2)

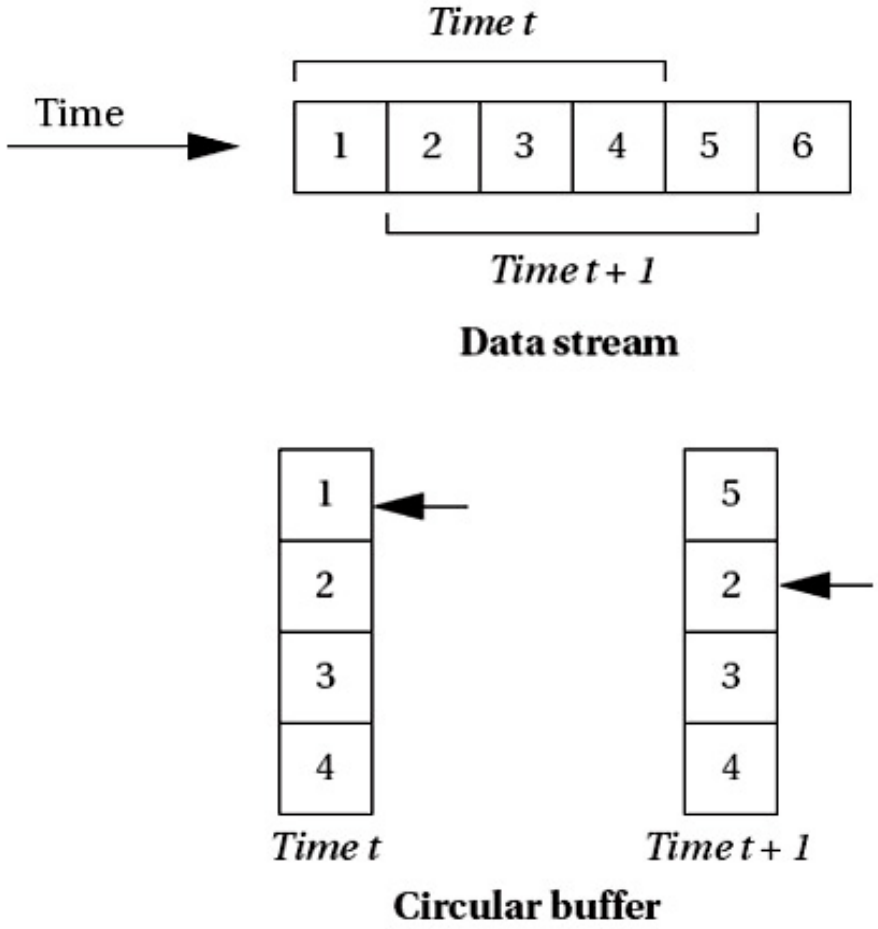


State machines are suited for reactive systems.

Design Patterns – State Machine (2/2)

```
switch (state) { /* check the current state */
  case IDLE:
    if (seat) { state = SEATED; timer_on = TRUE;}
    /* default case is self-loop */
    break;
  case SEATED:
    if (belt) state = BELTED; /* won't hear the buzzer */
     else if (timer) state = BUZZER; /* didn't put on belt in time */
    /* default is self-loop */
    break;
  case BELTED:
    if (!seat) state = IDLE; /* person left */
     else if (!belt) state = SEATED; /* person still in seat */
    break;
  case BUZZER:
    if (belt) state = BELTED; /* belt is on--turn off buzzer */
     else if (!seat) state = IDLE; /* no one in seat--turn off buzzer */
    break;
}
```

Design Pattern – Circular Buffer



- Real-time data processing (e.g. FIR) with limited memory.

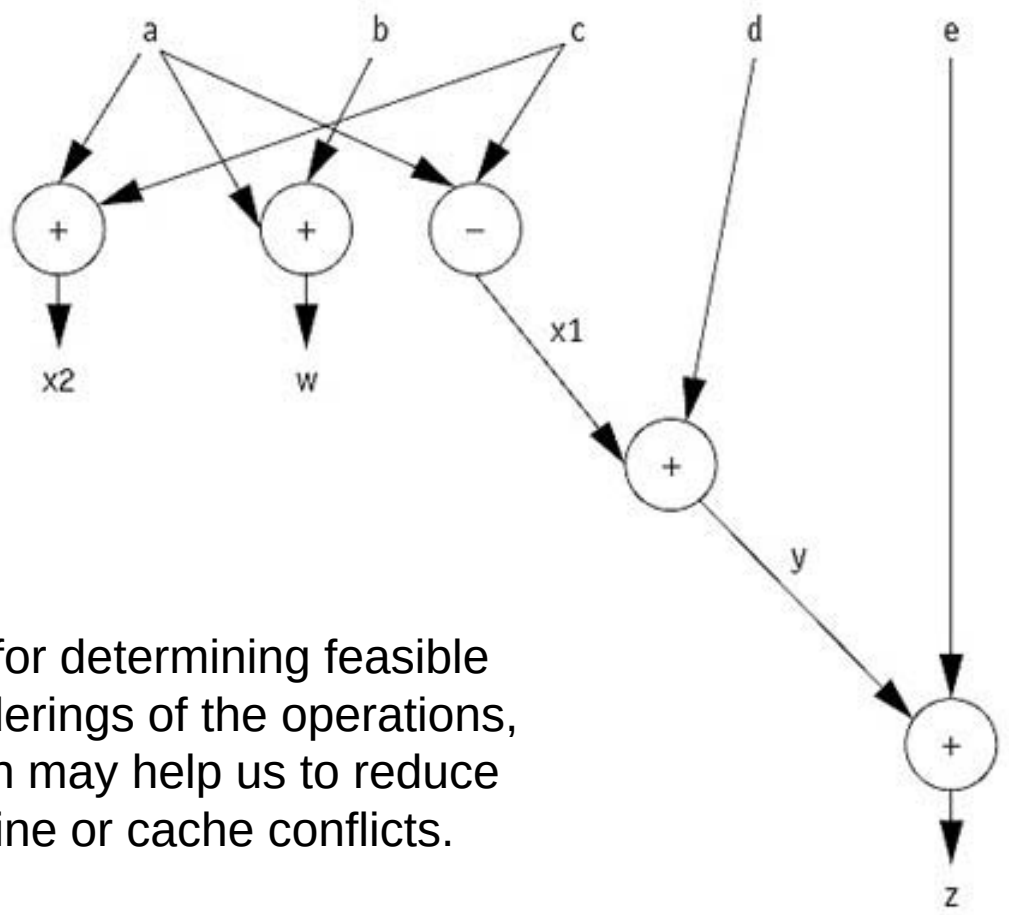
```
int ibuff, /*loop index for the circular buffer */
    ic; /* loop index for the coefficient array */
for (f =0, ibuff = circ_buff_head, ic=0; ic <N;
        ibuff = (ibuff == (N-1) ? 0 : ibuff++), ic++)
f =f + c[ic] * circ_buff[ibuff];
```

Models of Programs – Data Flow Graph

```
w = a + b;
x = a - c;
y = x+ d;
x = a + c;
z= y + e;
```

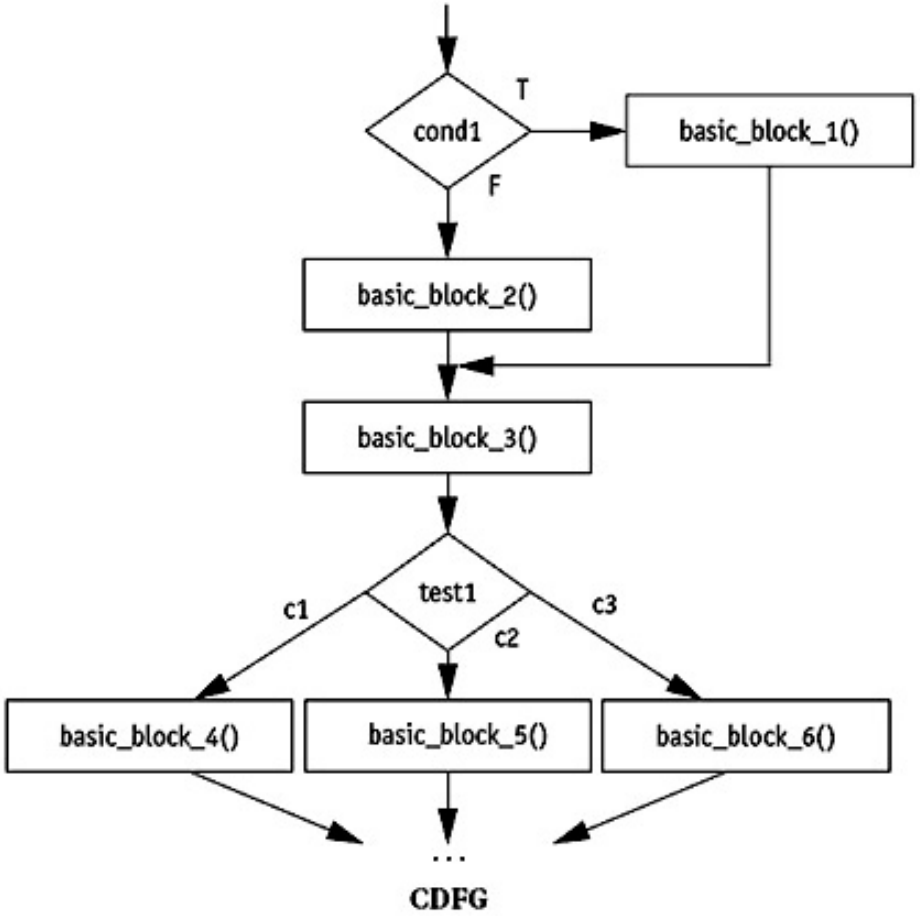
Single Assignment Form

```
w = a + b;
x1 = a - c;
y = x1 + d;
x2 = a + c;
z = y + e;
```



Use for determining feasible reorderings of the operations, which may help us to reduce pipeline or cache conflicts.

Control/Data Flow Graphs (CDFG)



Two types of nodes:
(1) Decision nodes
(2) Data flow nodes

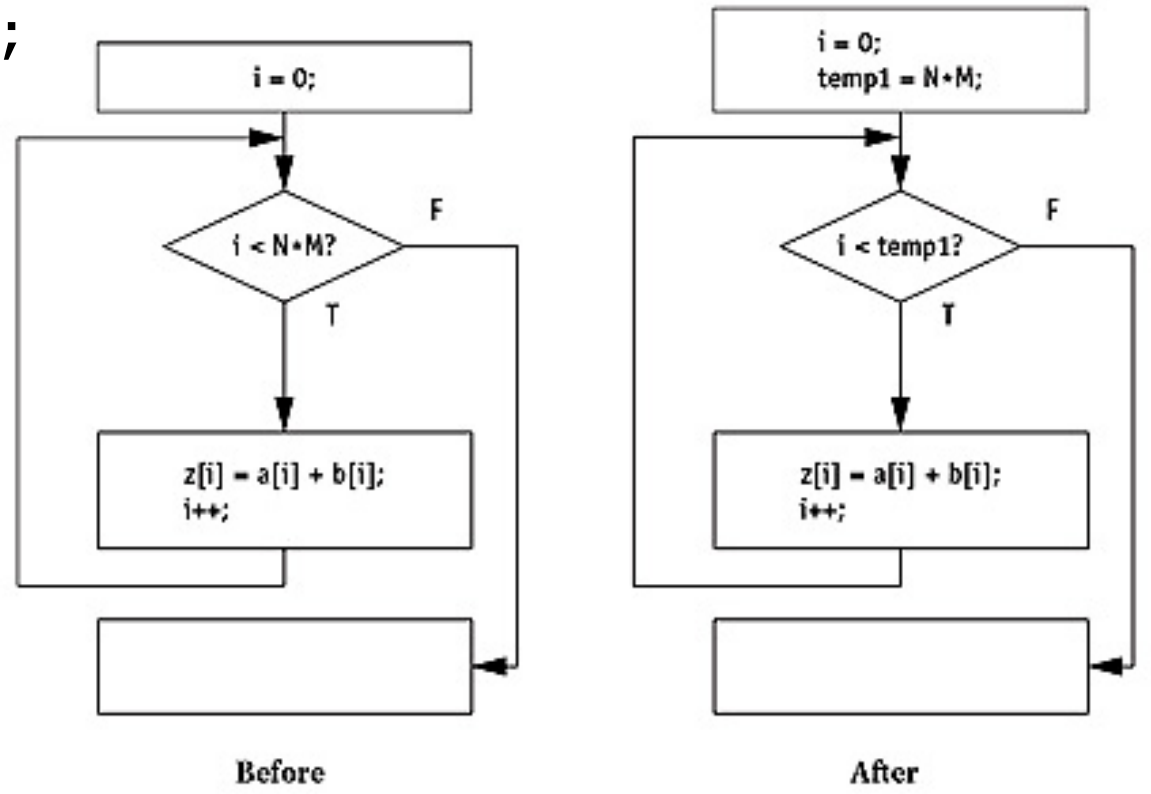
- The CDFG is a sequential representation of the program.

Measuring Execution Time

- method
  - simulator
    - provides greater visibility, profiling
    - not 100% accurate
  - timer
    - limited by the resolution of the timer
  - logic analyzer
- condition
  - average case, worst case, best case

Loop Optimizations – code motion

```
for (i = 0; i < N*M; i++) {
  z[i] = a[i] + b[i];
}
```



Move unnecessary code out of a loop.

Loop Optimizations – Induction Variable Elimination

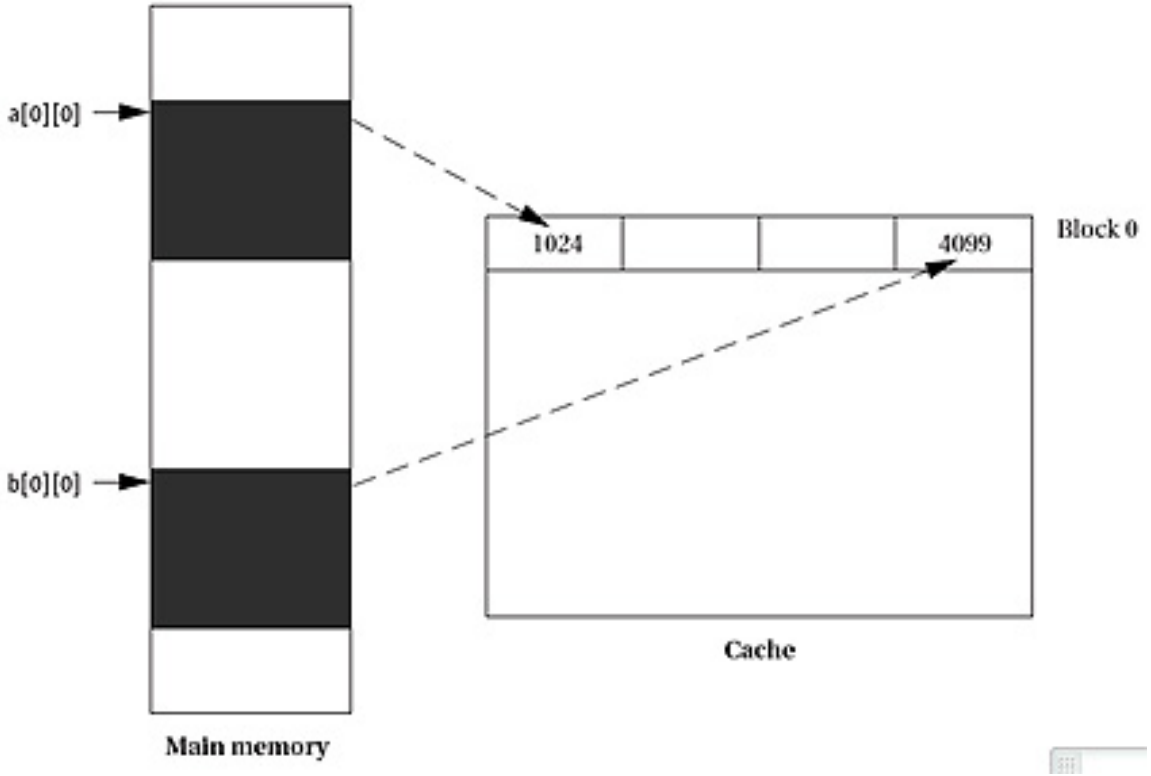
```
for (i = 0; i < N; i++)
  for (j = 0; j < M; j++)
    z[i][j] = b[i][j];
```


```
for (i = 0; i < N; i++)
  for (j = 0; j < M; j++) {
    zbinduct = i*M + j;
    *(zptr + zbinduct) = *(bptr + zbinduct);
}
```


```
for (i = 0; i < N; i++)
  for (j = 0; j < M; j++)
    *(zptr++) = *(bptr++);
```

Loop Nest – Cache Optimizations

```
for (j = 0; j < M; j++)
  for (i = 0; i < N; i++)
    a[j][i] = b[j][i] * c;
```

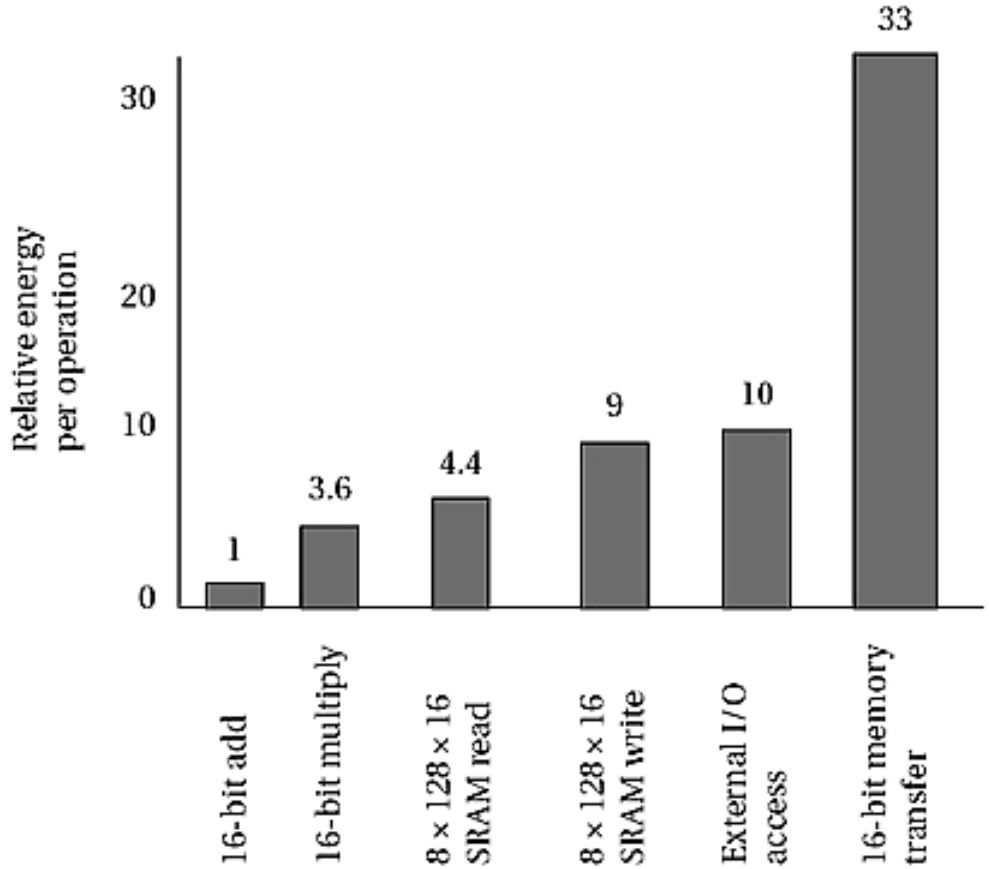


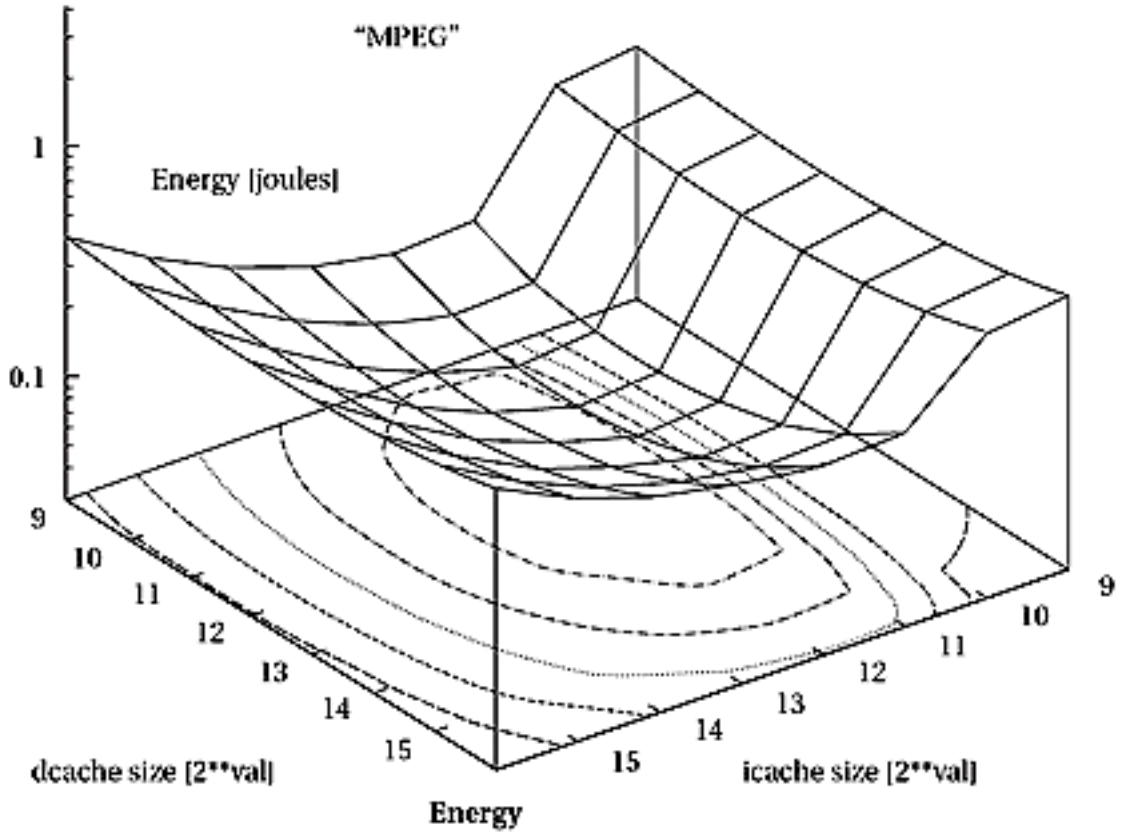
Problem:
Access *b* will replace *a*, and vice versa.

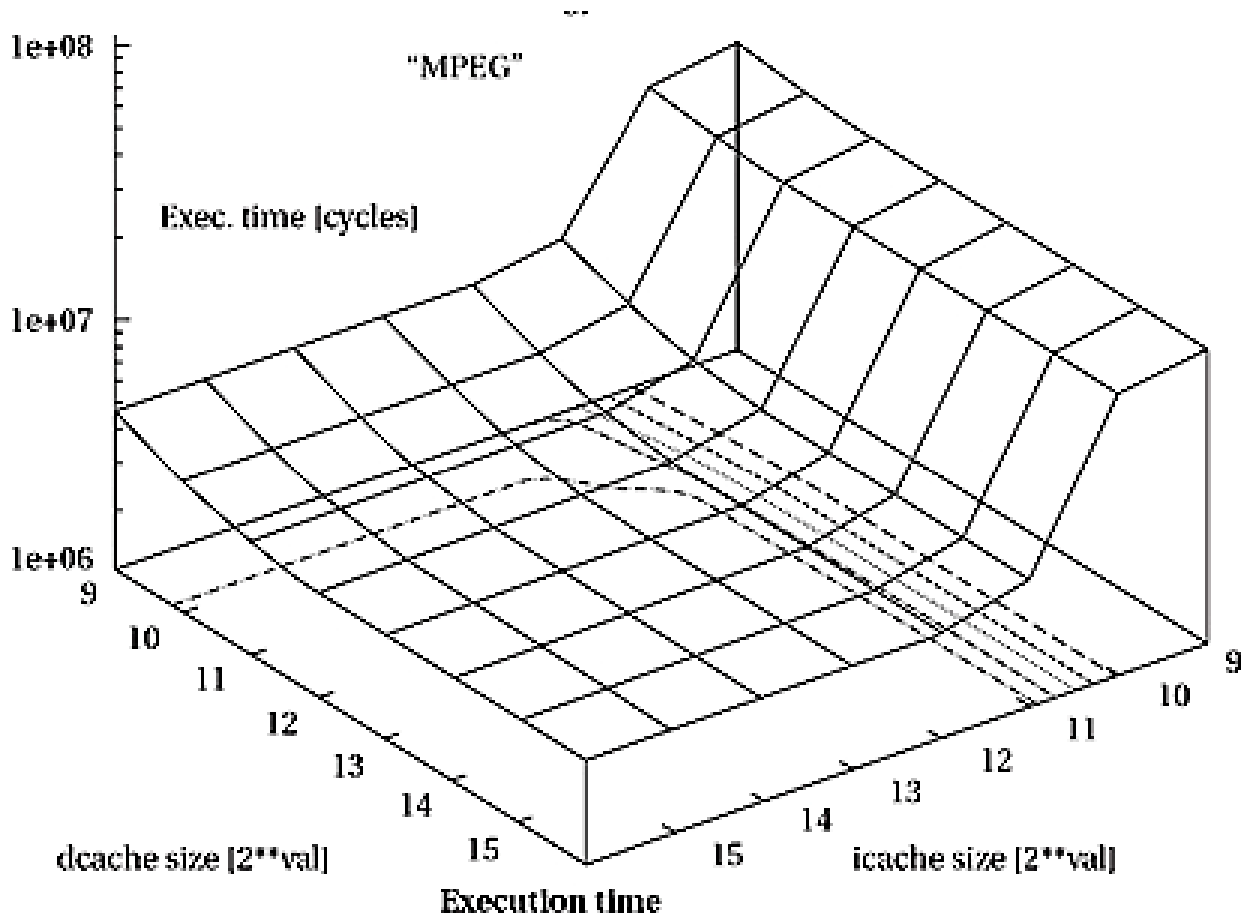
Solution:
(1) Data Realignment
(2) Array Padding

Power Saving Techniques

- Replace the algorithms with those that consume less power.
- Optimize memory access
  - memory accesses are a major component of power consumption in many applications.
- Turn off parts of the system when we don't need them
  - such as subsystems of the CPU, chips in the system


Relative energy per operation
0
10
20
30
1
3.6
4.4
9
10
33
16-bit add
16-bit multiply
8 × 128 × 16 SRAM read
8 × 128 × 16 SRAM write
External I/O access
16-bit memory transfer


"MPEG"
Energy [joules]
1
0.1
9
10
11
12
13
14
15
dcache size [2**val]
9
10
11
12
13
14
15
icache size [2**val]
Energy

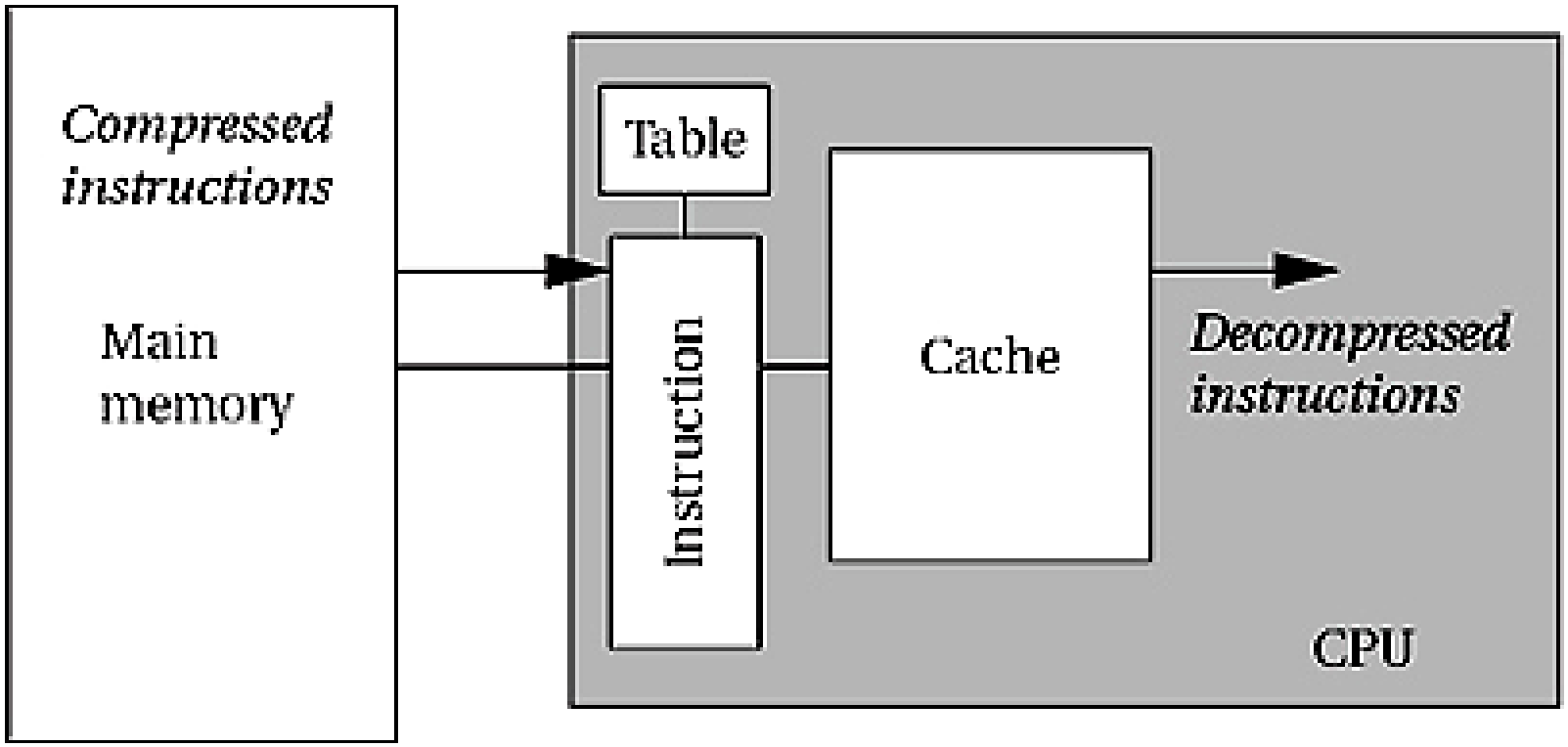

"MPEG"
Exec. time [cycles]
1e+08
1e+07
1e+06
9
10
11
12
13
14
15
dcache size [2**val]
9
10
11
12
13
14
15
icache size [2**val]
Execution time

Reducing Data Size

- Identifying and eliminating duplications of data
- Determine the actual maximum amount of data held in the buffer and allocate the array accordingly
- Data can sometimes be packed
- Reuse values
- Generate data on the fly

Reducing Code Size

- Encapsulating functions in subroutines can reduce program size when done carefully
  - Proper use of subroutines: e.g. performs identical operations repeatedly
- Change sequence of instructions
  - for example, a multiply-accumulate instruction may be both smaller and faster than separate arithmetic operations.
- Use dense instruction sets
  - e.g. ARM Thumb 16-bit instruction set vs. ARM 32-bit
- Compress the code and decompress on the fly


Compressed instructions
Main memory
Table
Instruction
Cache
Decompressed instructions
CPU

Testing

- Black Box Test
- White Box Test
- Random Test
- Regression Test
- Coverage Test
- Boundary Test
- Performance Test

Code coverage of functional tests for TeX and awk (after Horgan and Mathur [Hor96])

	Block	Decision	P-use	C-use
TeX	85%	72%	53%	48%
awk	70%	59%	48%	55%

Reference

- The basics of programming embedded processors, Wayne Wolf, Embedded.com.
- "Computers as Components: Principles of Embedded Computer System Design“, Wayne Wolf.