

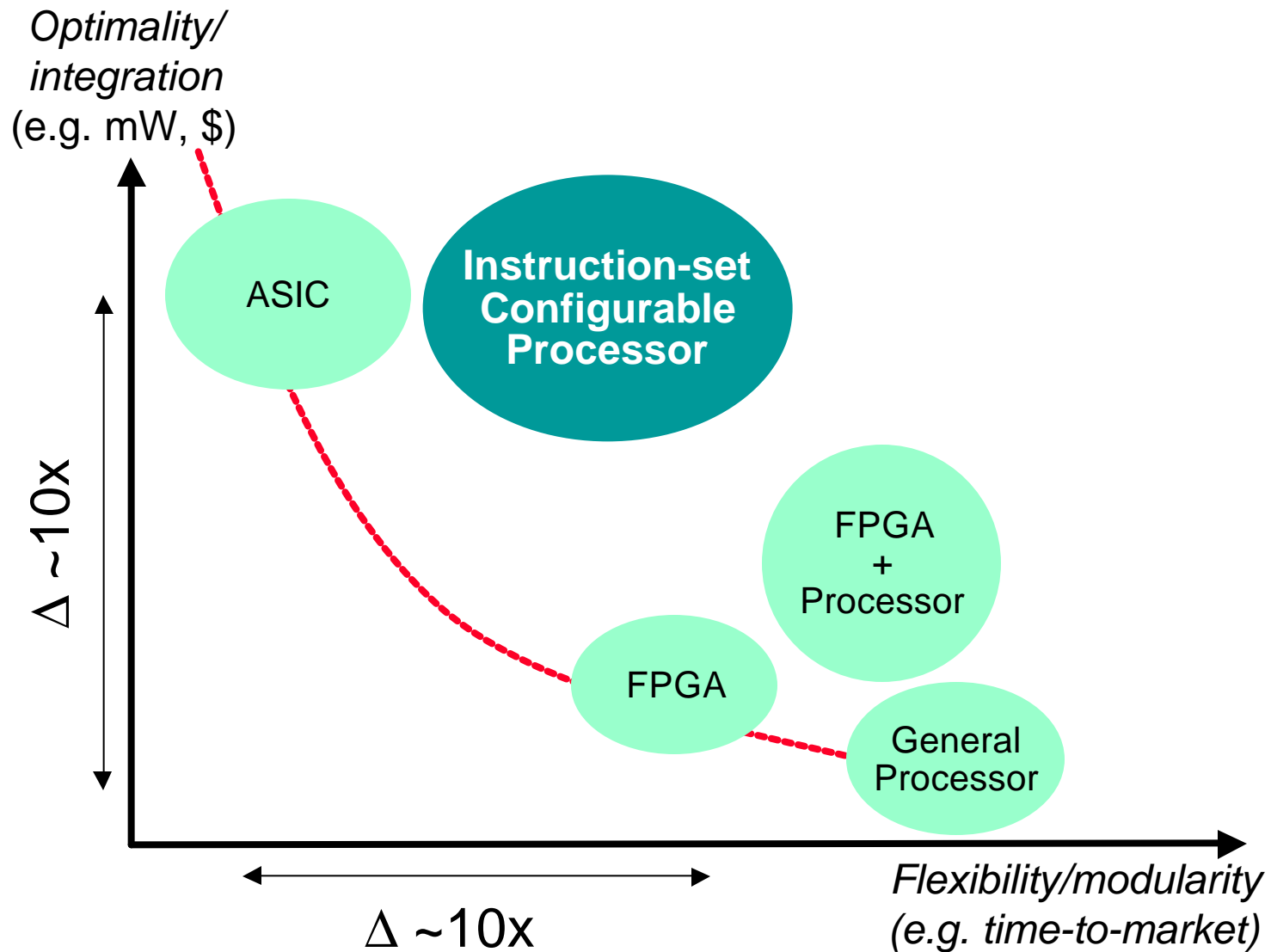
Hardware/Software Instruction Set Configurability for System-on-Chip Processors



Albert Wang, Chris Rowen,
Dror Maydan, Earl Killia

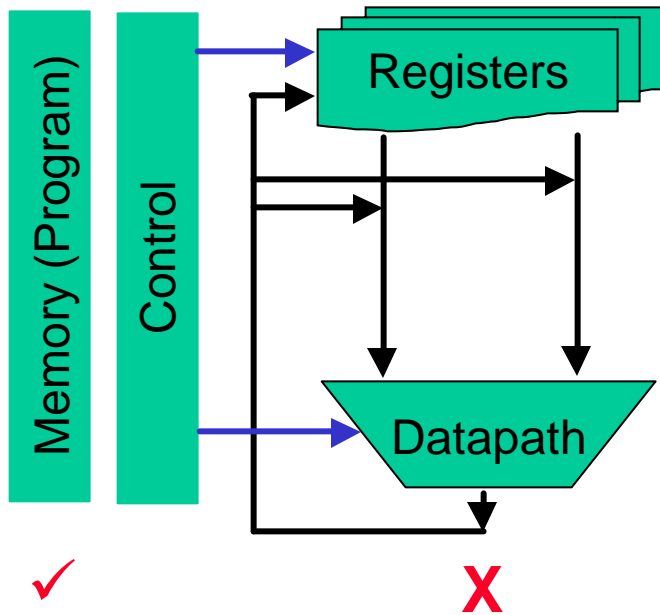
38th DAC, Las Vegas,
June 18-22, 2001

Landscape of reconfigurable computing



Computing using temporal connection

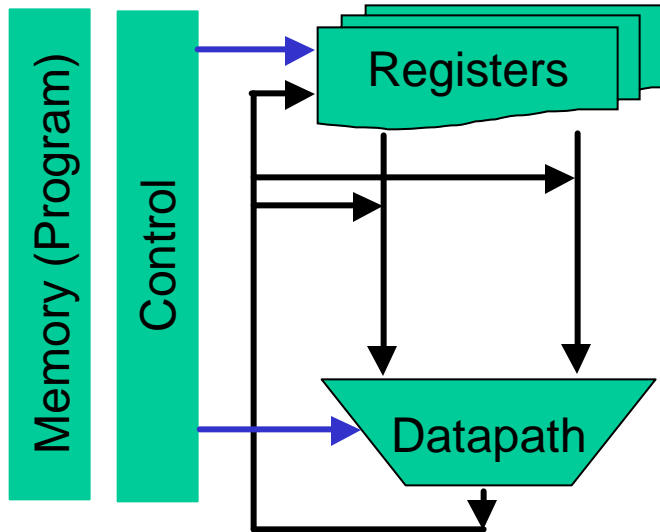
Processor Solution



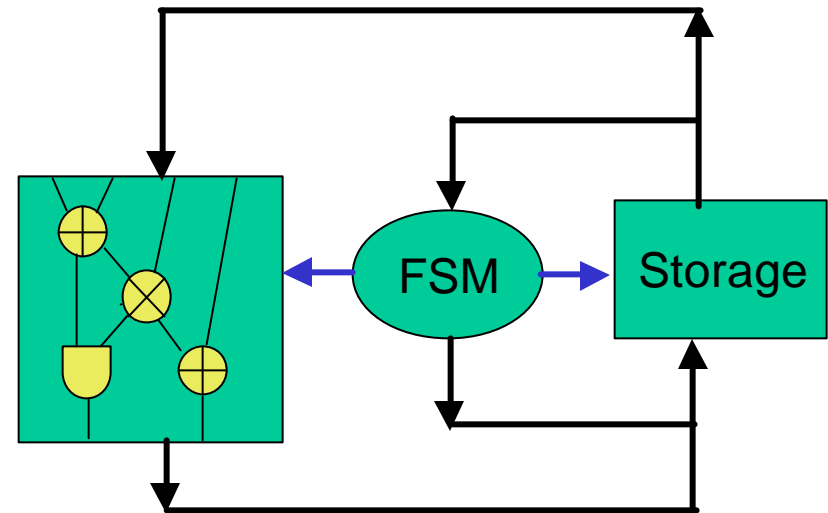
	Correct	Efficient
Processor	✓	X

Computing using spatial connection

Processor Solution



ASIC Solution



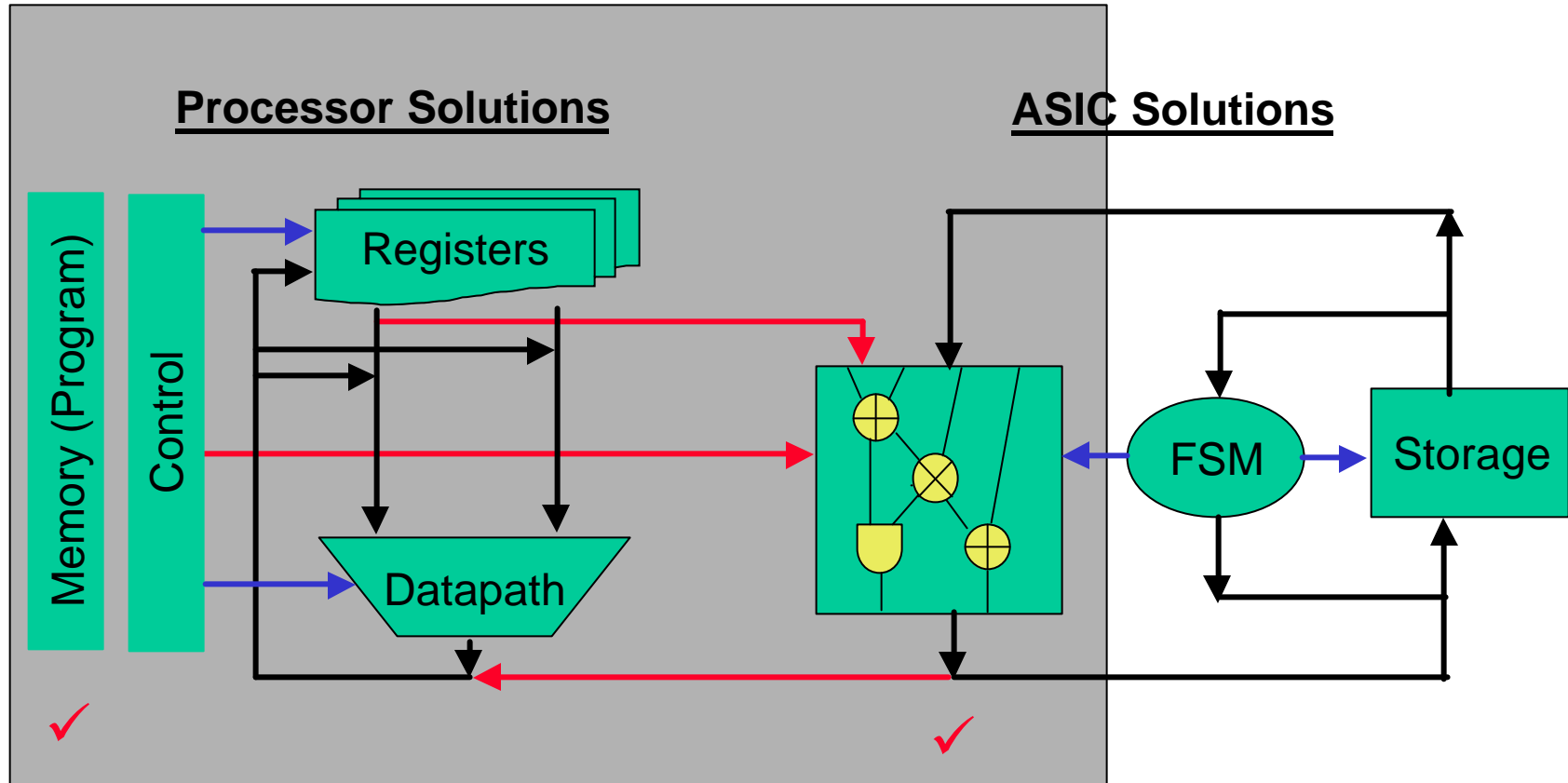
✓

✗

	Correct	Efficient
ASIC	✗	✓

Configurable Processors: best of both

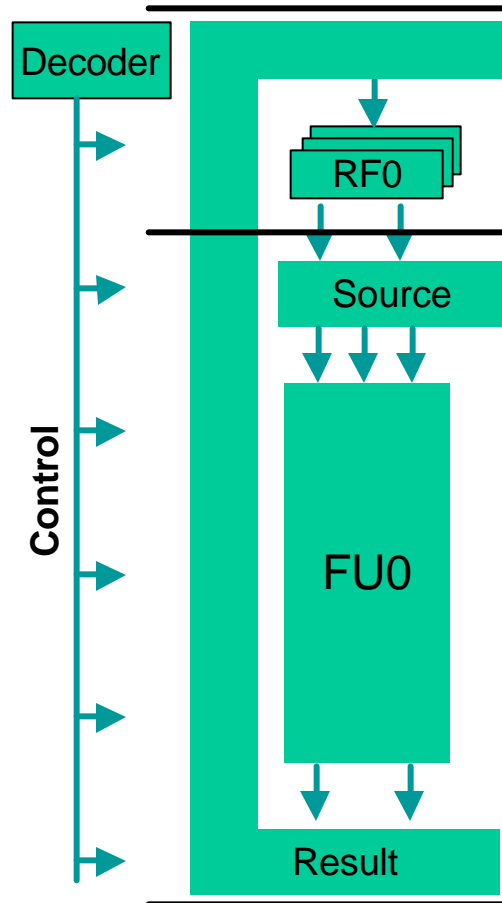
Processor with Application-specific Instructions



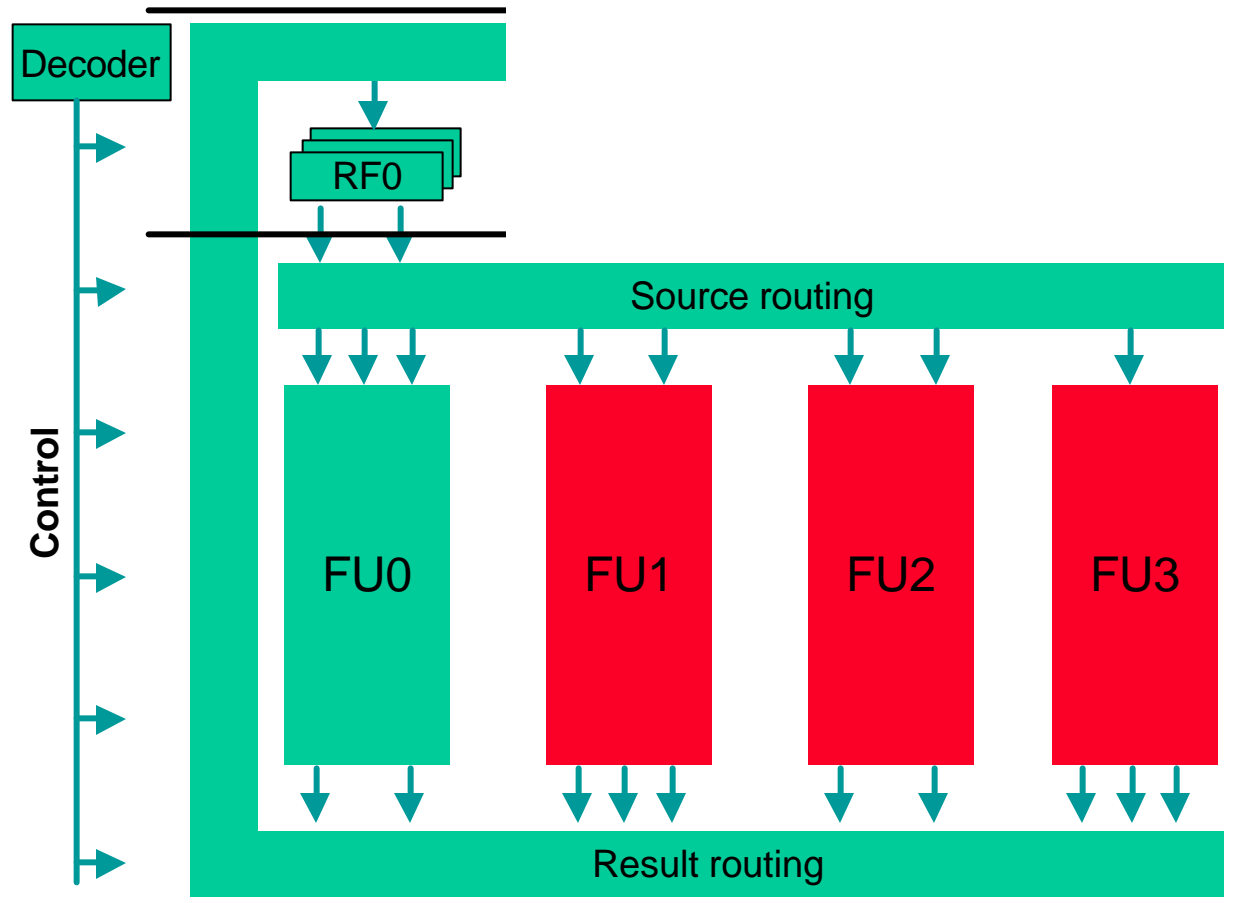
	Correct	Efficient
Processor	✓	
ASIC		✓

- ❖ **Configurable processor solution**
 - **Xtensa™ processor Architecture**
 - **Instruction extension automation**
 - **Software development tools**
- ❖ **An Example**
- ❖ **Results**
- ❖ **Summary**

Conventional Architecture

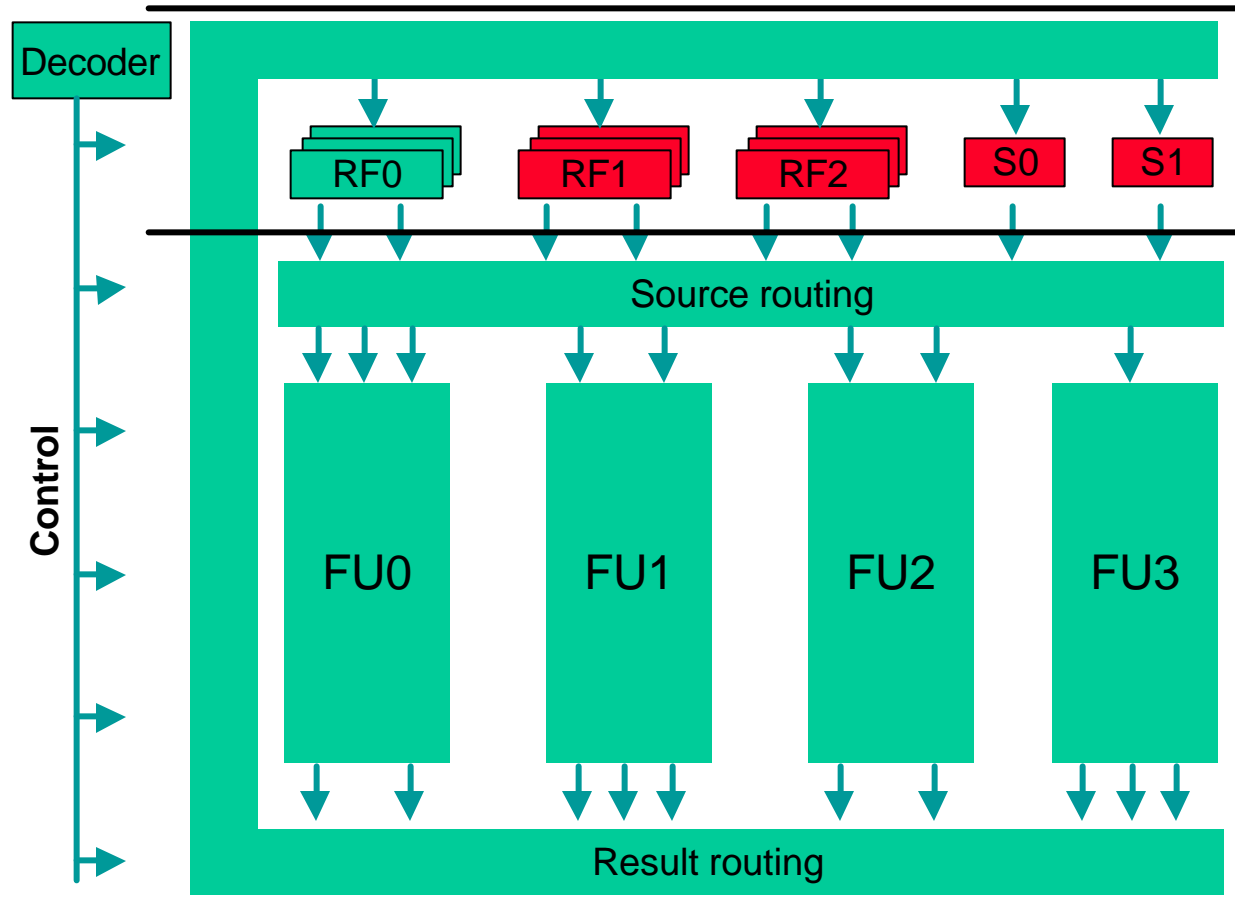


Conventional Architecture - cont.



•More FU's

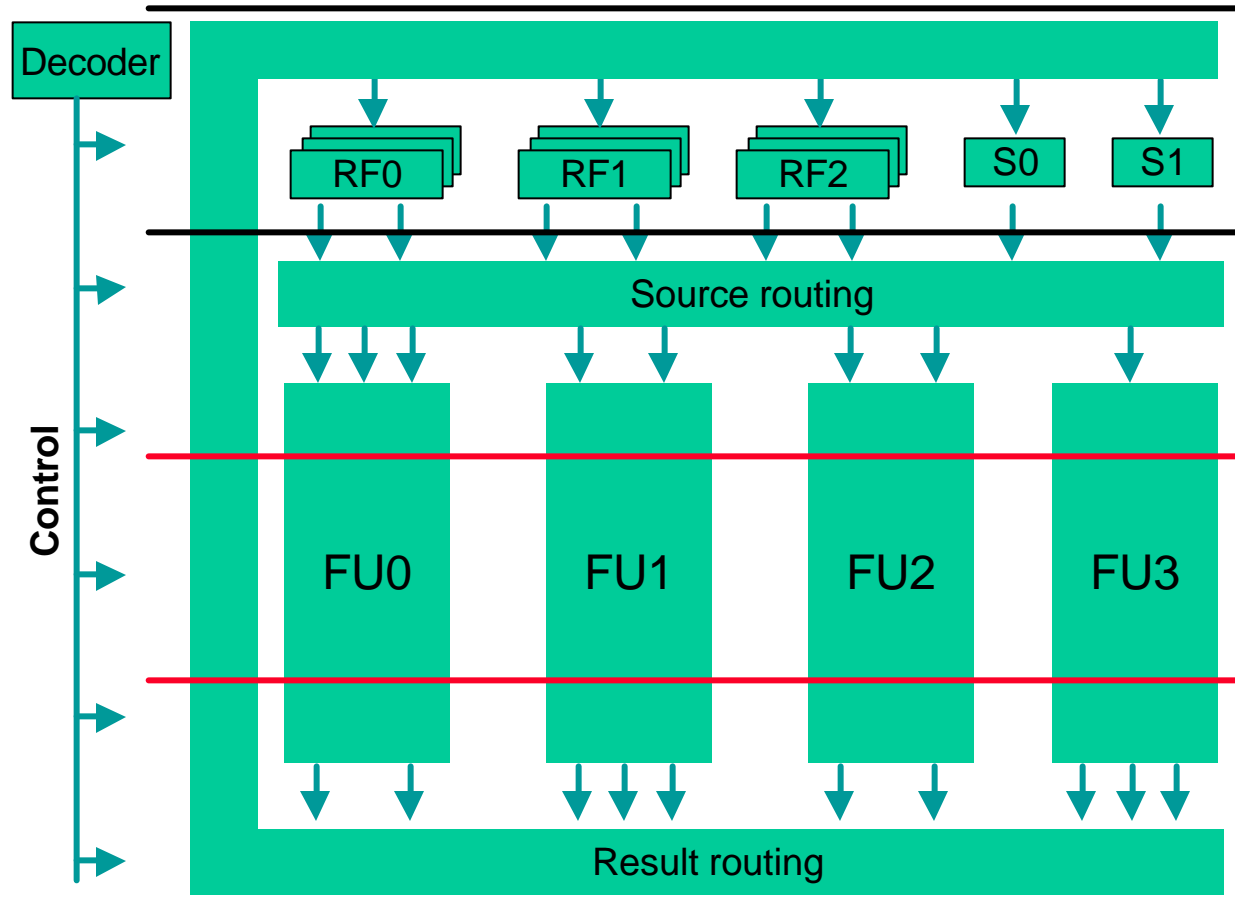
Conventional Architecture – cont.



- More FU's

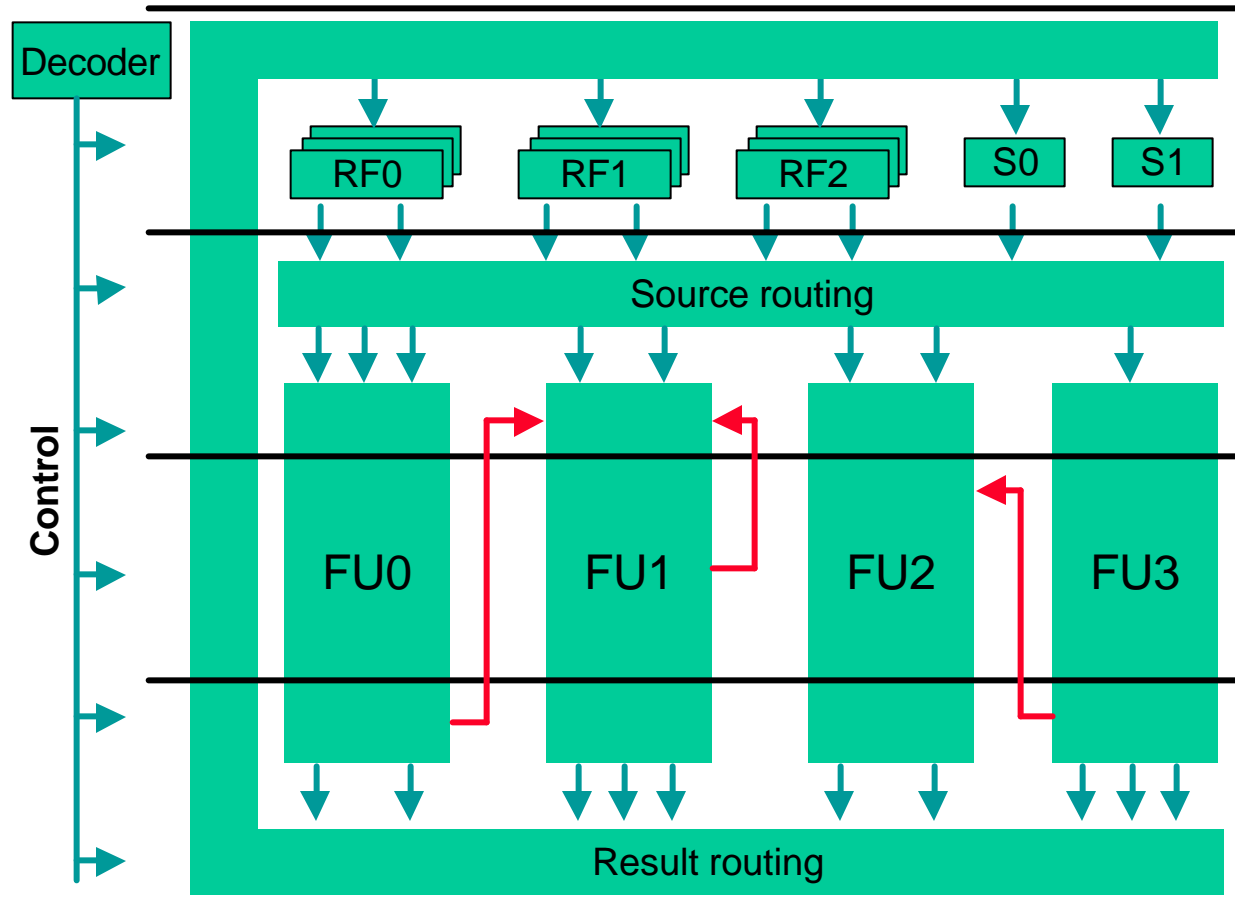
- More registers

Conventional Architecture – cont.



- More registers
- More FU's
- **Deeper pipeline**

Conventional Architecture – cont.



- More registers
- More FU's
- Deeper pipeline
- **Bypass/forward**

Conventional Architecture – cont.

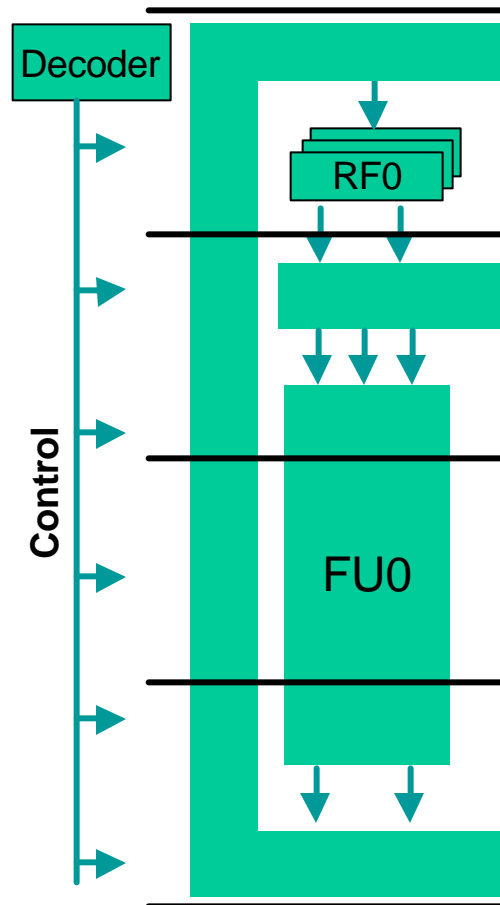
❖ Problem with fixed processor:

- Waste silicon
 - There is no universal extensions, or even one for each application class
- Not fast enough, compared with hardware implementation
- Waste power

❖ The Tensilica solution:

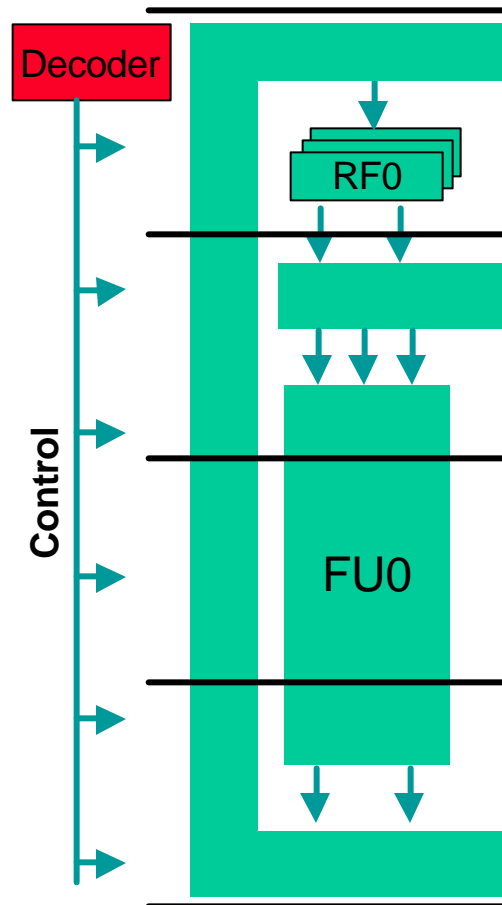
- Small core processor
- Allow **easy** and **efficient** application-specific instruction extensions

Xtensa Architecture – Base



- ❖ **Good performance**
 - Comparable to any embedded 32-bit RISC
- ❖ **Good code density**
 - Much better than 32-bit RISC
 - Use 16b/24b instructions
- ❖ **Small**
 - .7mm² in .18
- ❖ **Low power**
 - .37mw / MHz
- ❖ **Easy extension**
 - With Tensilica Instruction Extension (TIE) language – ISA level
- ❖ **Efficient extension**
 - TIE compiler generates efficient pipelined implementation
 - TIE compiler extends all software development tools

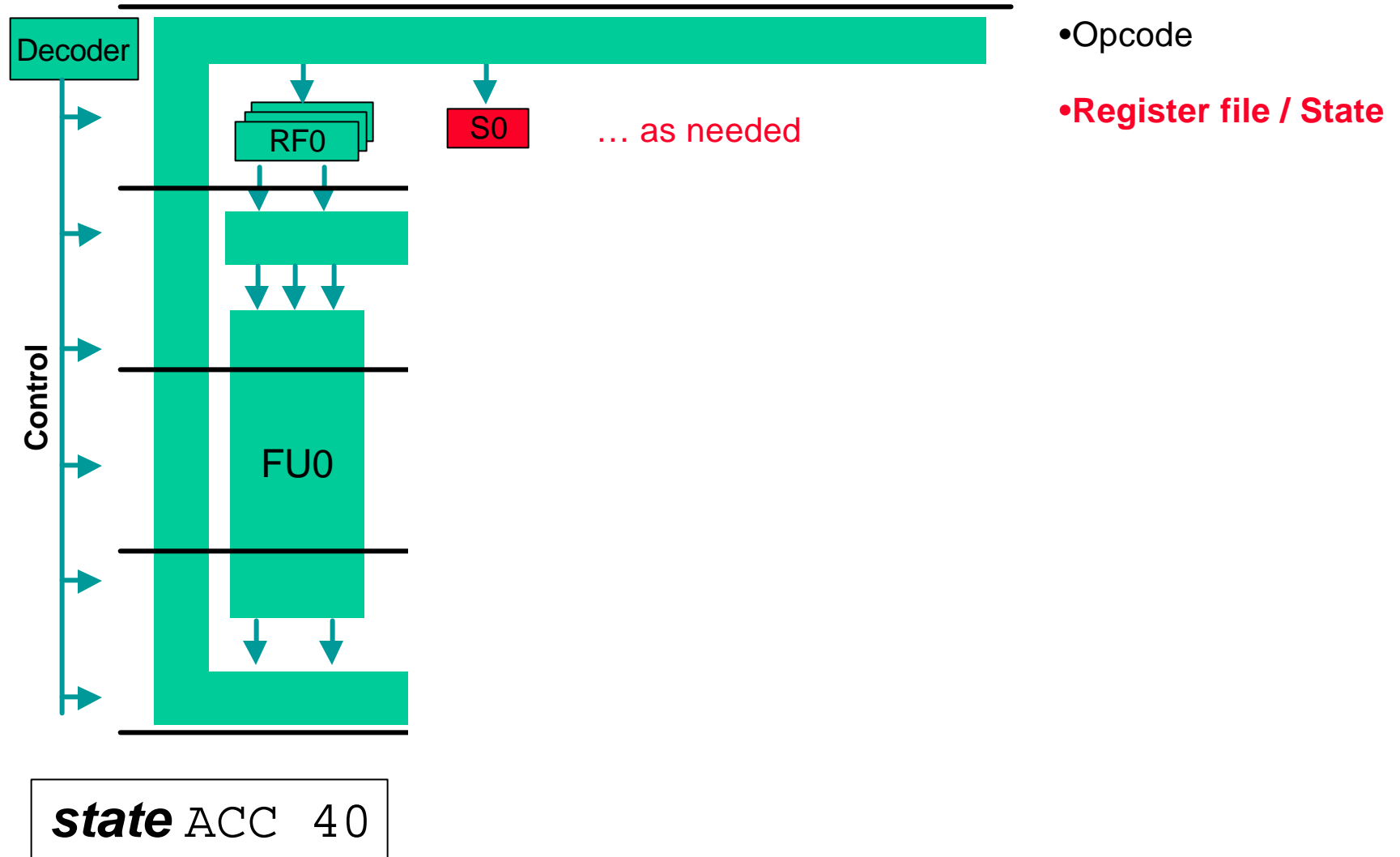
TIE language - opcode



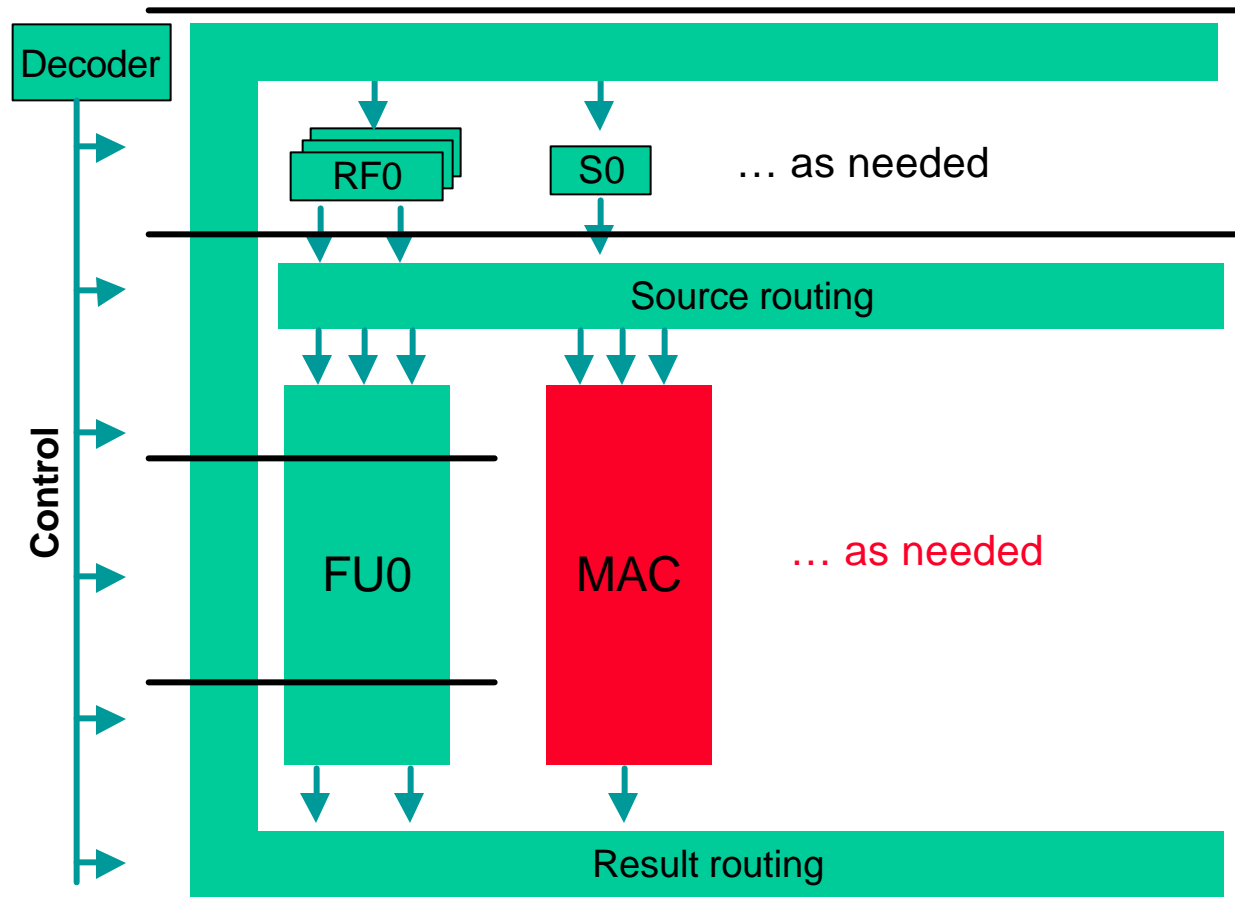
•Opcode

opcode MAC *op2=5 CUST0*

TIE Language - regfile / state



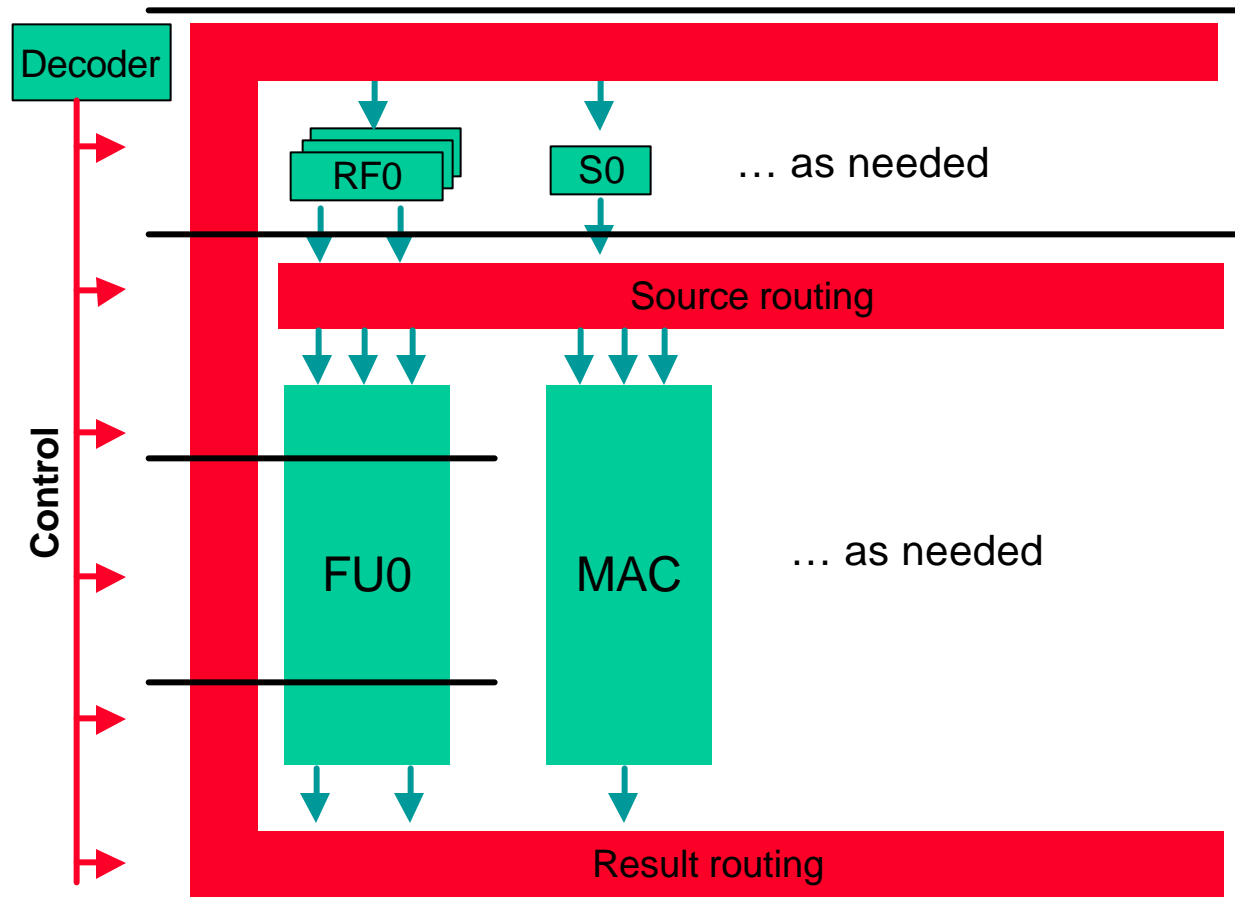
TIE Language - semantics



- Opcode
- Register file / state
- **semantics**

```
semantic sem1 {MAC} {assign ACCL=ACCL+ars[16:0]*art[15:0];}
```

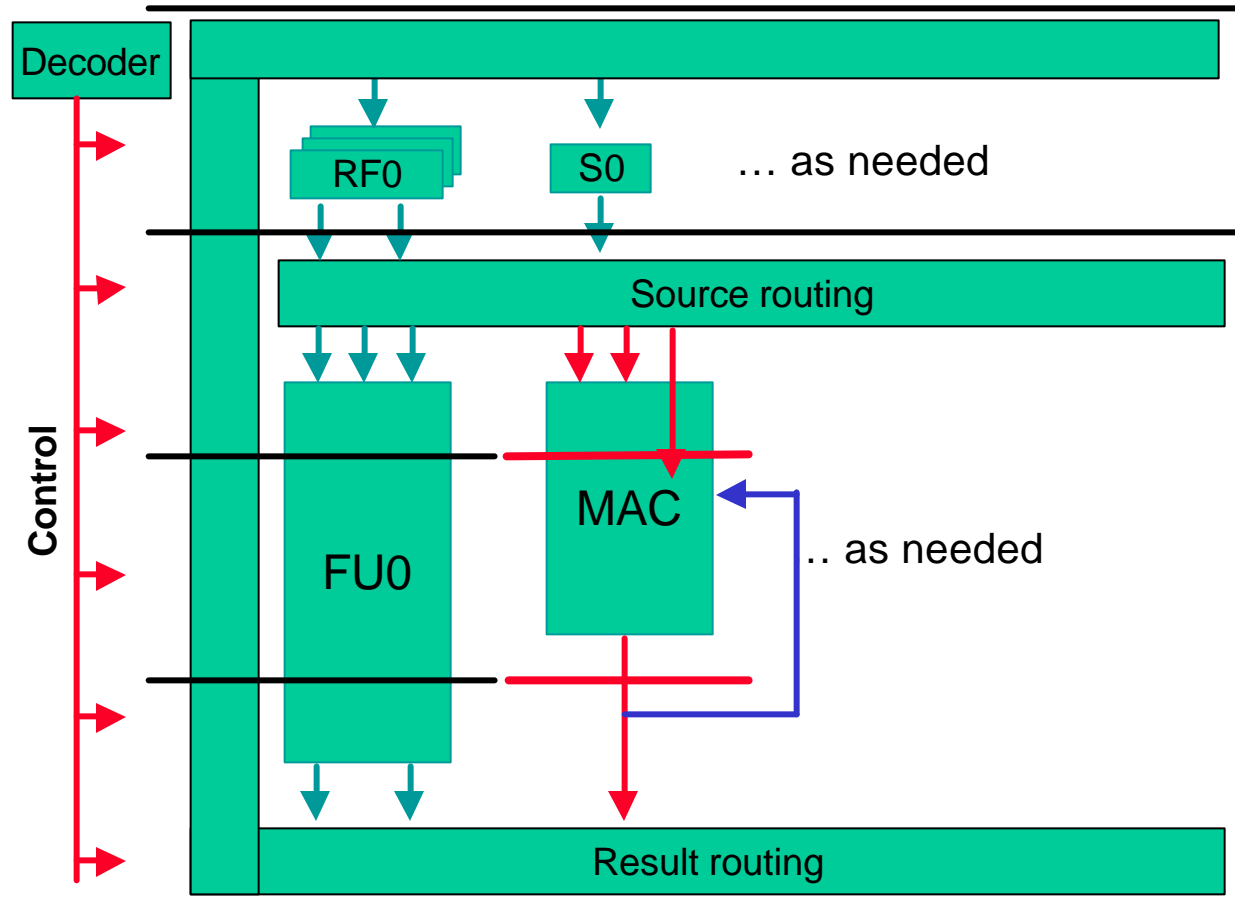
TIE Language - *iclass*



- Opcode
- Register file / state
- semantics
- **Instruction class**

```
iclass c1 {MAC} {in ars, in art} {inout ACC}
```

TIE Language - schedule



- Opcode
- Register file / state
- semantics
- **Instruction class**
- **schedule**

```
schedule s1 {MAC}{use ars 1; use art 1; use ACC 2; def ACC 2;}
```

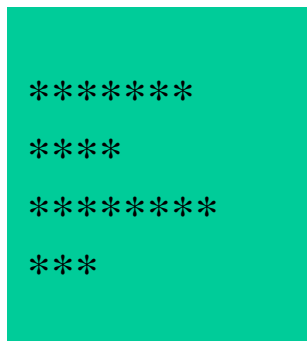
A Complete Example – parallel MAC

```
opcode PMAC op2=0 CUST0
state ACC1 40
state ACC2 40
iclass rr {PMAC}{in ars, in art}{inout ACC1, inout ACC2}
semantic pmac_sem {PMAC} {
    assign ACC1 = ACC1 + ars[15:0] * art[15:0];
    assign ACC2 = ACC2 + ars[31:16] * art[31:16];
}
schedule pmac_schd {PMAC} {
    use ars 1; use art 1;
    use ACC1 2; use ACC2 2;
    def ACC1 2; def ACC2 2;
}
```

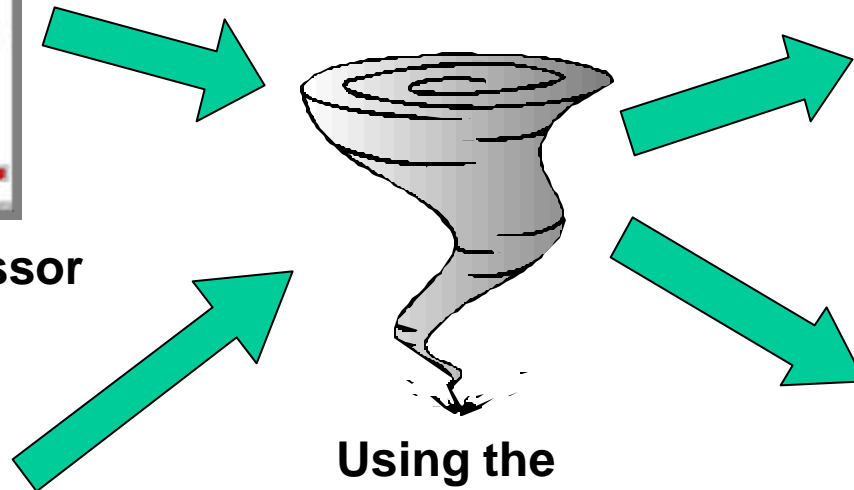
Productivity Gain - language + compiler



Select processor options

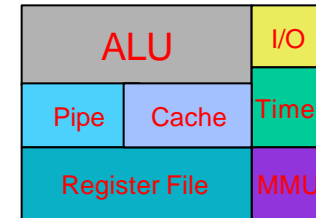


Describe new instructions

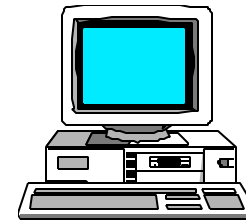


Using the **Xtensa** processor generator, create...

In Minutes!



Tailored, synthesizable HDL uP core

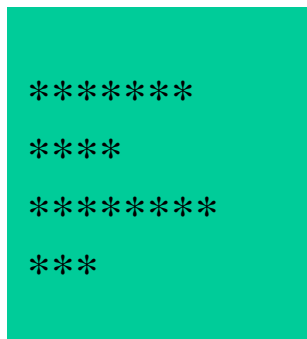


Customized Compiler, Assembler, Linker, Debugger, Simulator

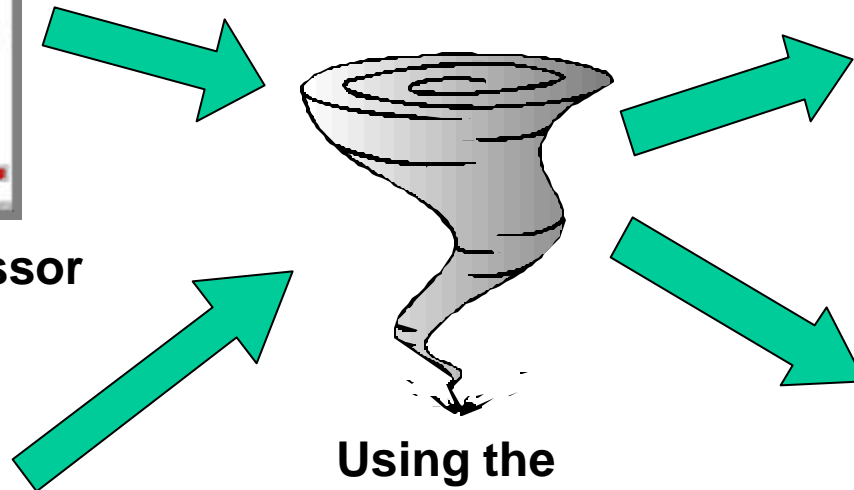
Productivity Gain – Software Tools



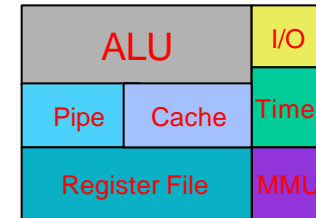
Select processor options



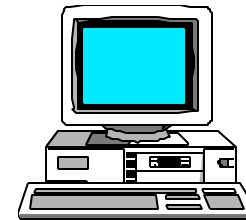
Describe new instructions



Using the **Xtensa** processor generator, create...

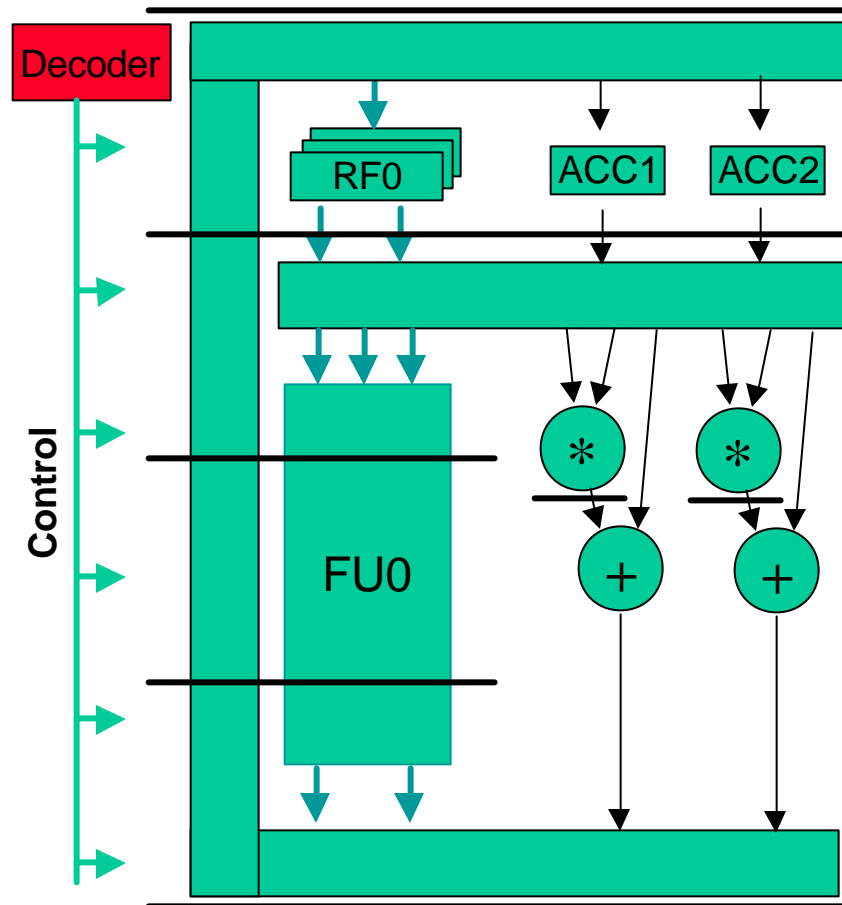


Tailored, synthesizable HDL uP core



Customized Compiler, Assembler, Linker, Debugger, Simulator

Software Support – Assembler

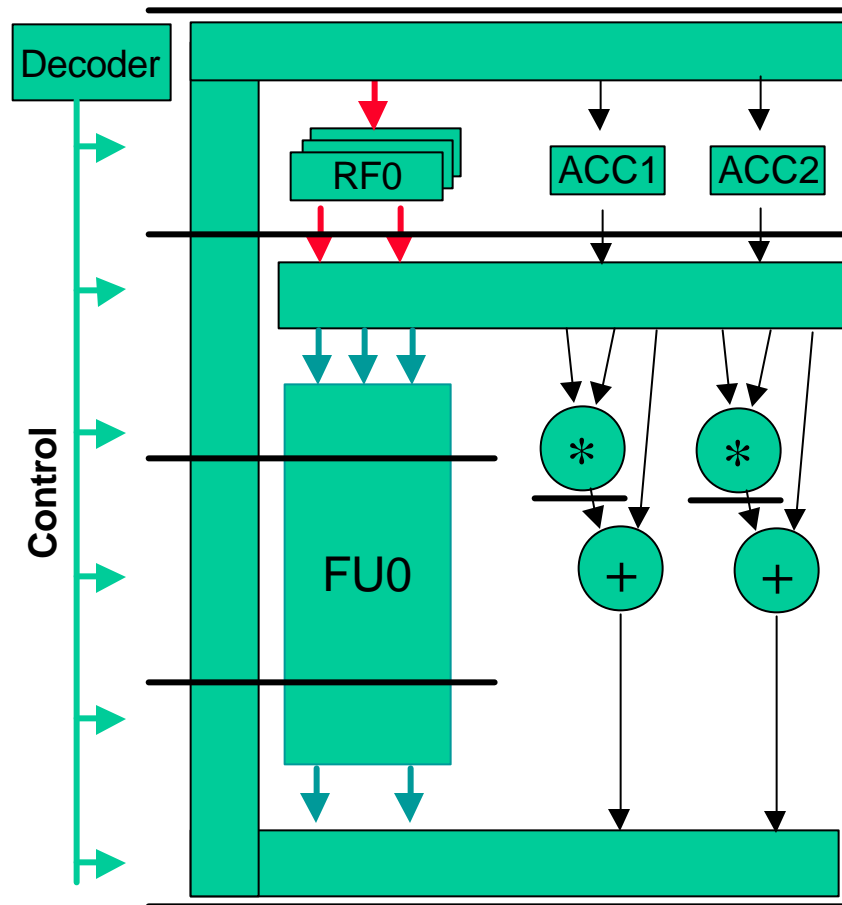


•Assembler

```
Loop a2, .L1
  l16si  a10, a3, 0
  l16si  a11, a3, 2
  addi.n a3, a3, 2
  PMAC   a10, a11
.L1:
```

- Custom data type
- Register allocation
- Code Scheduling
- RTOS
- Simulator/debugger

Software Support – custom data type



•Assembler

•Custom data type

```
C Code: sat_int x,y,z;  
z = sat_add(x,y);
```

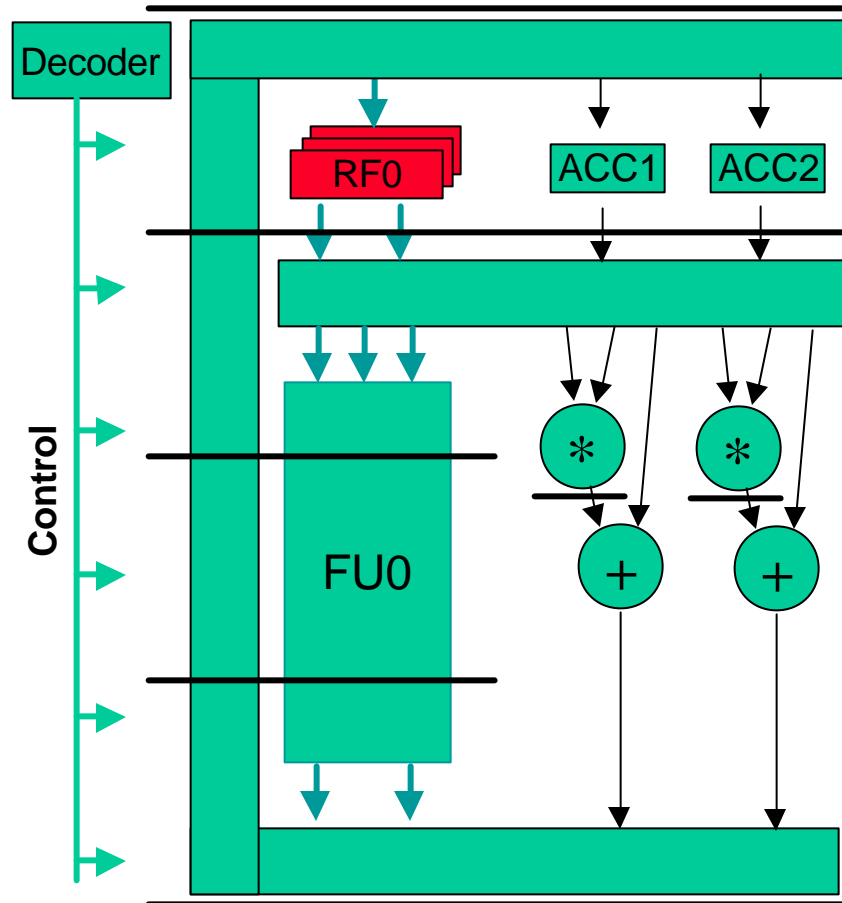
•Register allocation

•Code Scheduling

•RTOS

•Simulator/debugger

Software Support – register allocation



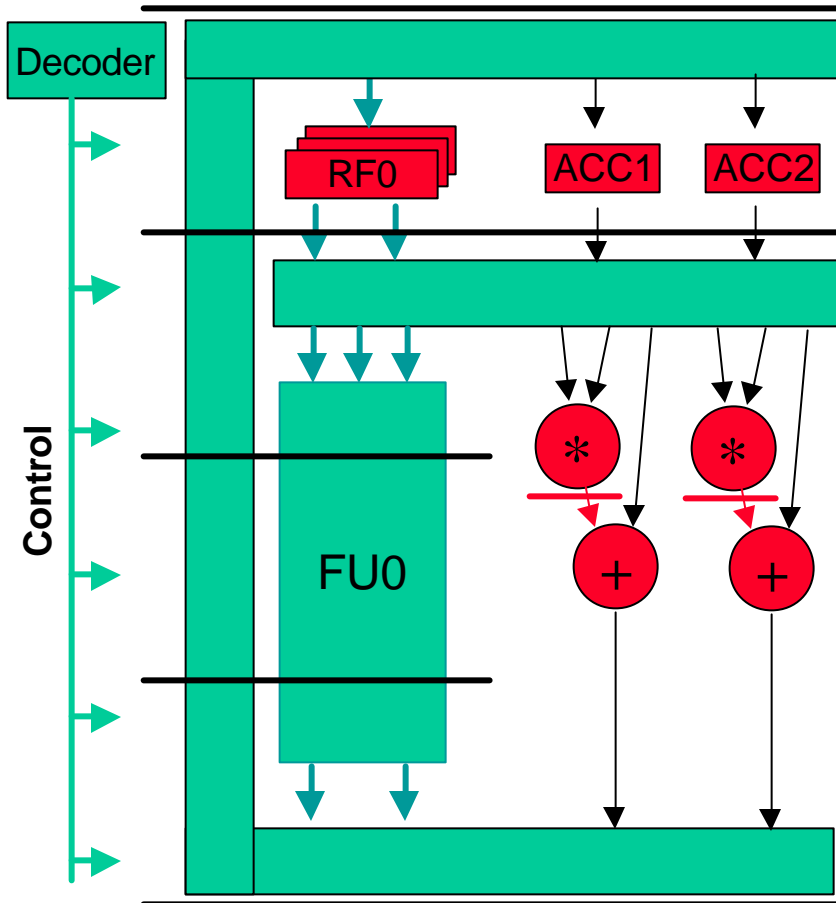
- Assembler
- Custom data type
- Register allocation**

Spilling around a call:

```
sat_add    s3, s1, s2
sat_store  s3, a1, 0
call18    foo
sat_load   s3, a1, 0
```

- Code Scheduling
- RTOS
- Simulator/debugger

Software Support – code scheduling



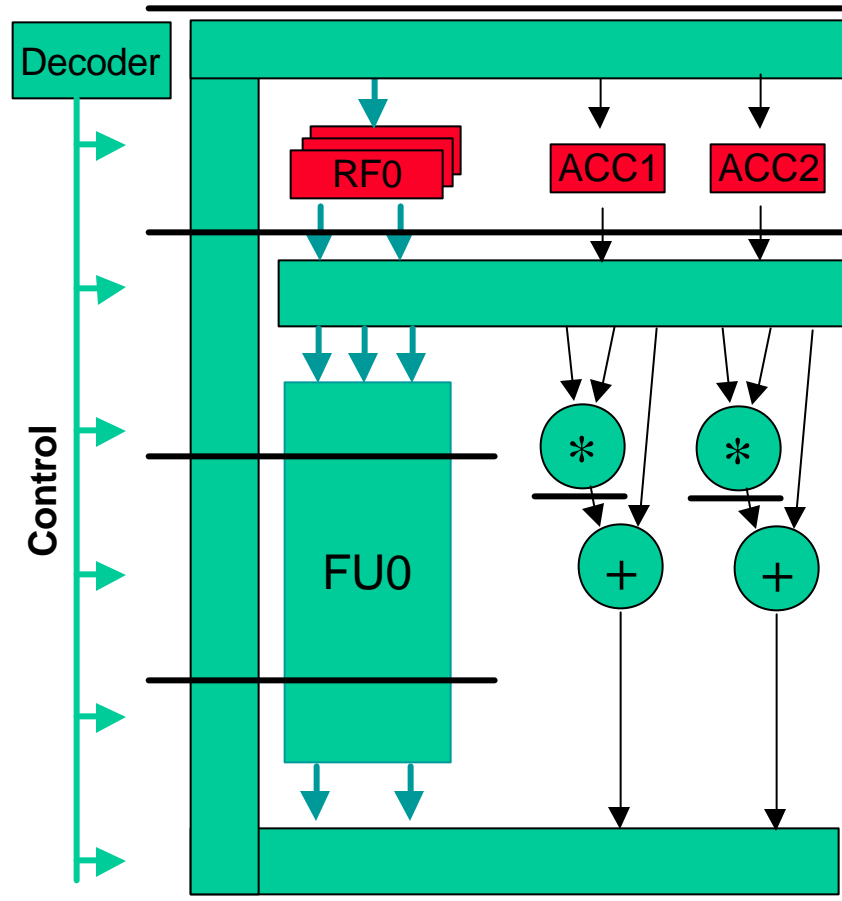
- Assembler
- Custom data type
- Register allocation
- Code Scheduling

```
t = sat_mult(x,y);  
z = sat_add(z, t);  
t2 = sat_mult(x2, y2);
```

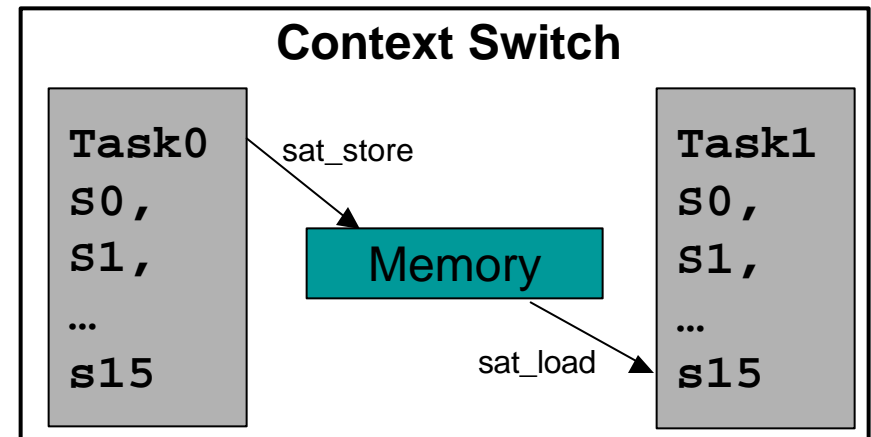
```
sat_mult s3, s1, s2  
sat_mult s6, s5, s4  
sat_add s7, s7, s3
```

- RTOS
- Simulator/debugger

Software Support - RTOS

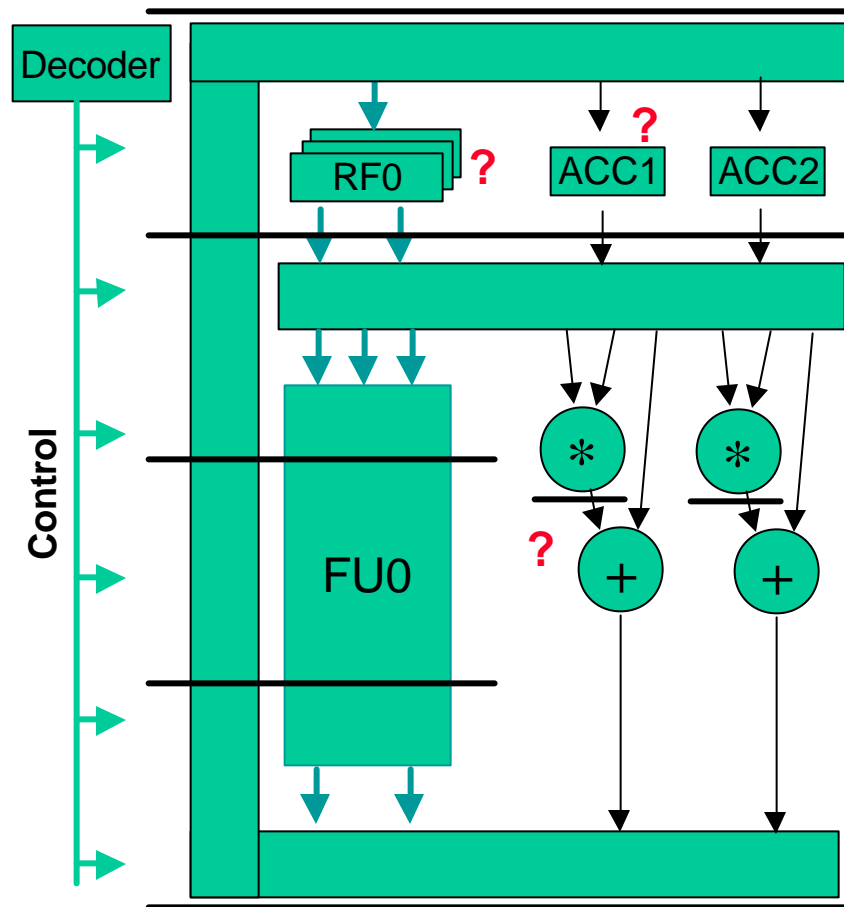


- Assembler
- Custom data type
- Register allocation
- Code Scheduling
- RTOS



- Simulator/debugger

Software Support – simulator/debugger



- Assembler
- Custom data type
- Register allocation
- Code Scheduling
- RTOS
- Simulator/debugger**

```
gdb> break ...
```

```
gdb> cont
```

```
gdb> step
```

```
gdb> display ...
```

❖ Configurable processors

- Architecture
- Instruction extension
- Software support

❖ An Example

❖ Results

❖ Summary

Data Encryption Standard (DES)

Initial step

$(R, L) = \text{Initial_permutation}(\text{Din}_{64})$

Iterate 16 times

Key generation

$(C, D) = \text{PC1}(k)$

$n = \text{rotate_amount}(\text{function of iteration count})$

$C = \text{rotate_right}(C, n)$

$D = \text{rotate_right}(D, n)$

$K = \text{PC2}(D, C)$

Encryption

$R_{i+1} = L_i \oplus \text{Permutation}(\text{S_Box}(K \oplus \text{Expansion}(R)))$

$L_{i+1} = R_i$

Final step

$\text{Dout}_{64} = \text{Final_permutation}(L, R)$

DES: Software Implementation

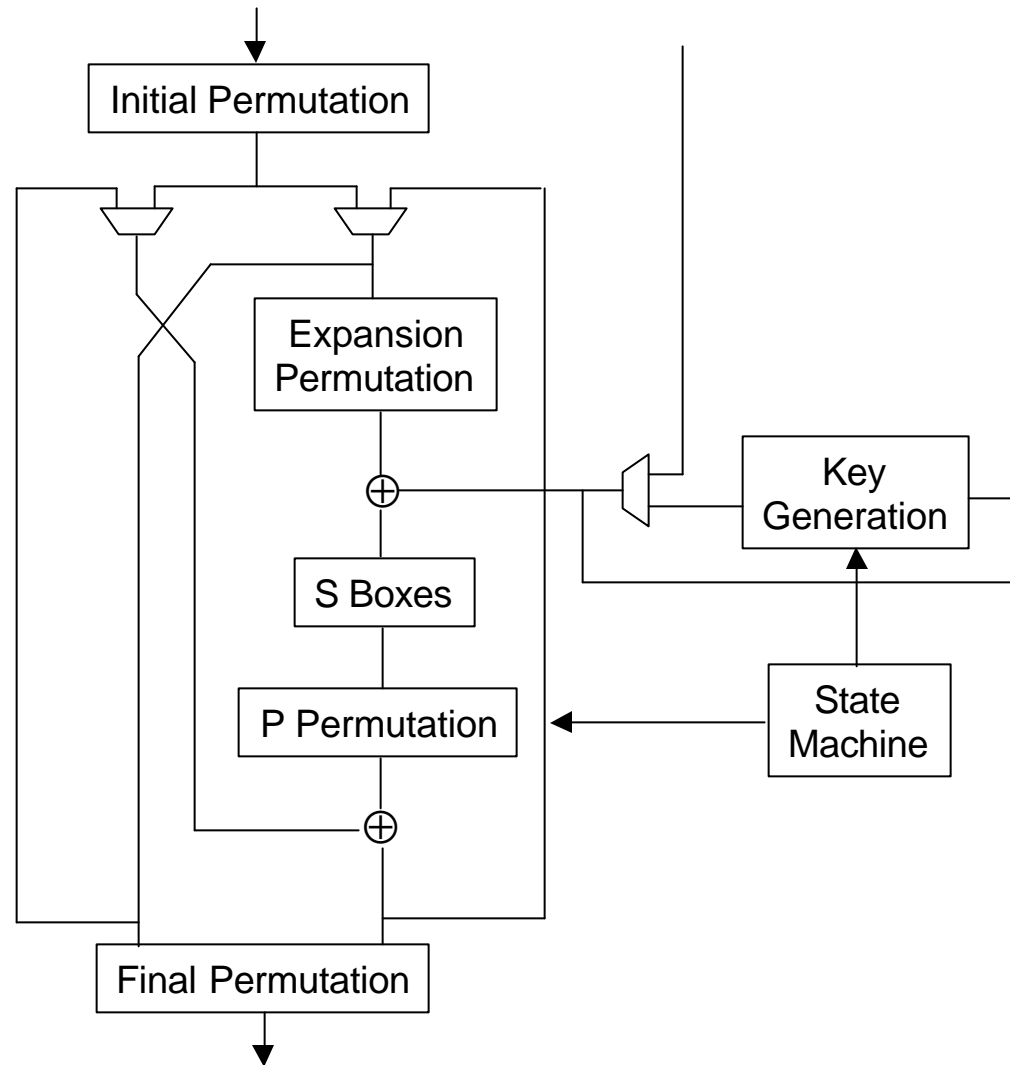
```
static unsigned permute(  
    unsigned char *table,  
    int n,  
    unsigned hi,  
    unsigned lo)  
{  
    int ib, ob;  
    unsigned out = 0;  
    for (ob = 0; ob < n; ob++) {  
        ib = table[ob] - 1;  
        if (ib >= 32) {  
            if (hi & (1 << (ib-32))) out |= 1 << ob;  
        } else {  
            if (lo & (1 << ib)) out |= 1 << ob;  
        }  
    }  
    return out;  
}
```

DES: Software Implementation

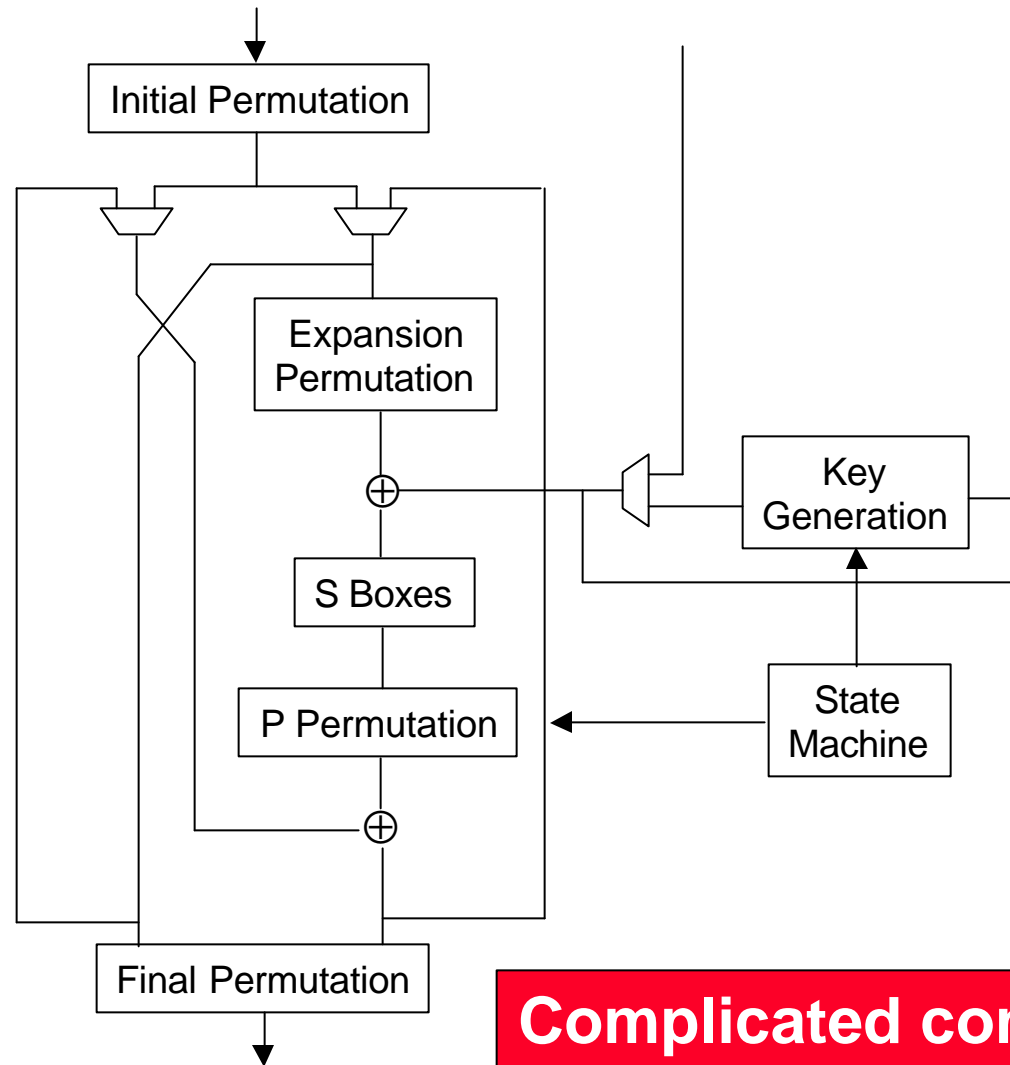
```
static unsigned permute(  
    unsigned char *table,  
    int n,  
    unsigned hi,  
    unsigned lo)  
{  
    int ib, ob;  
    unsigned out = 0;  
    for (ob = 0; ob < n; ob++) {  
        ib = table[ob] - 1;  
        if (ib >= 32) {  
            if (hi & (1 << (ib-32))) out |= 1 << ob;  
        } else {  
            if (lo & (1 << ib)) out |= 1 << ob;  
        }  
    }  
    return out;  
}
```

Too much computation!

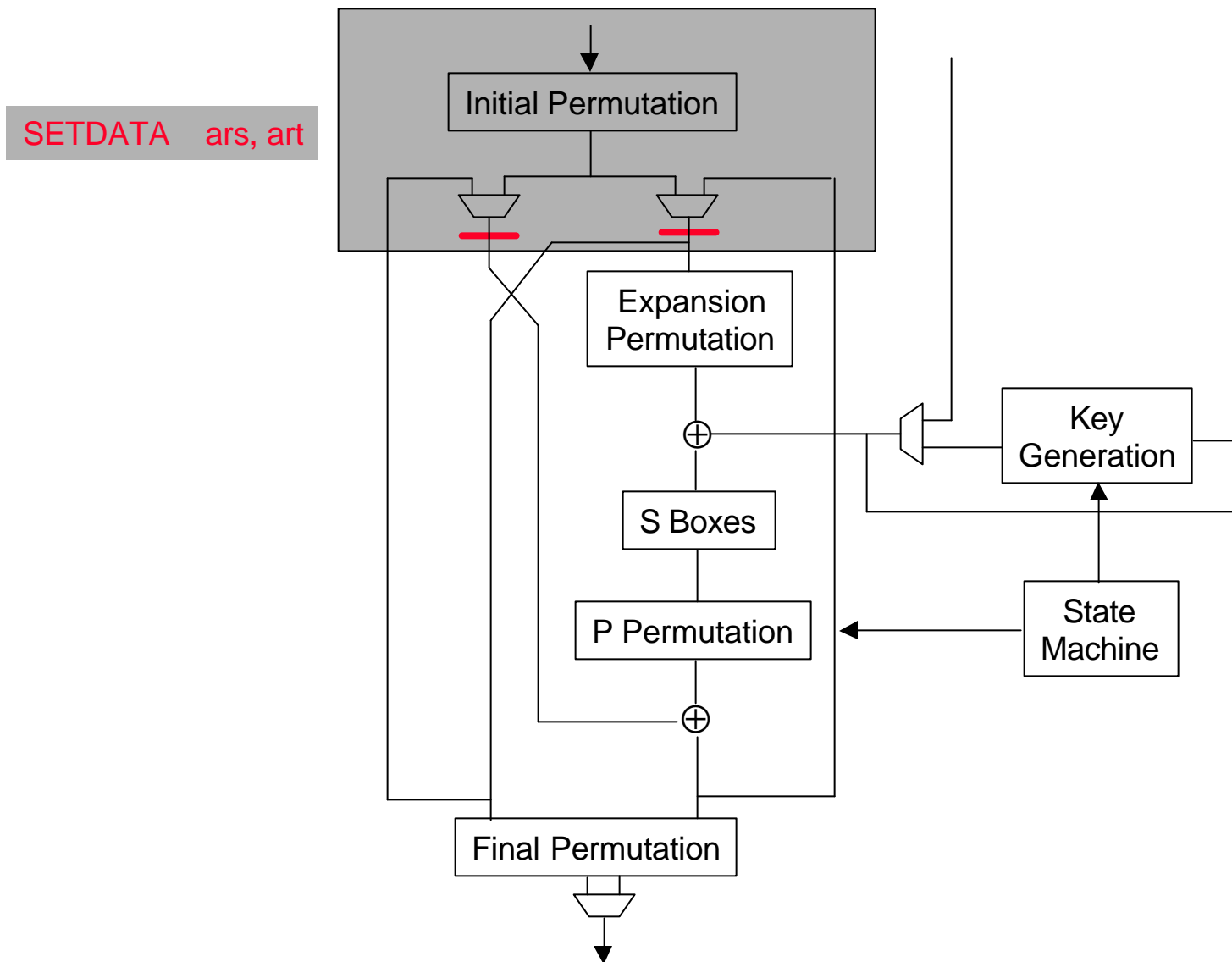
DES: Hardware Implementation



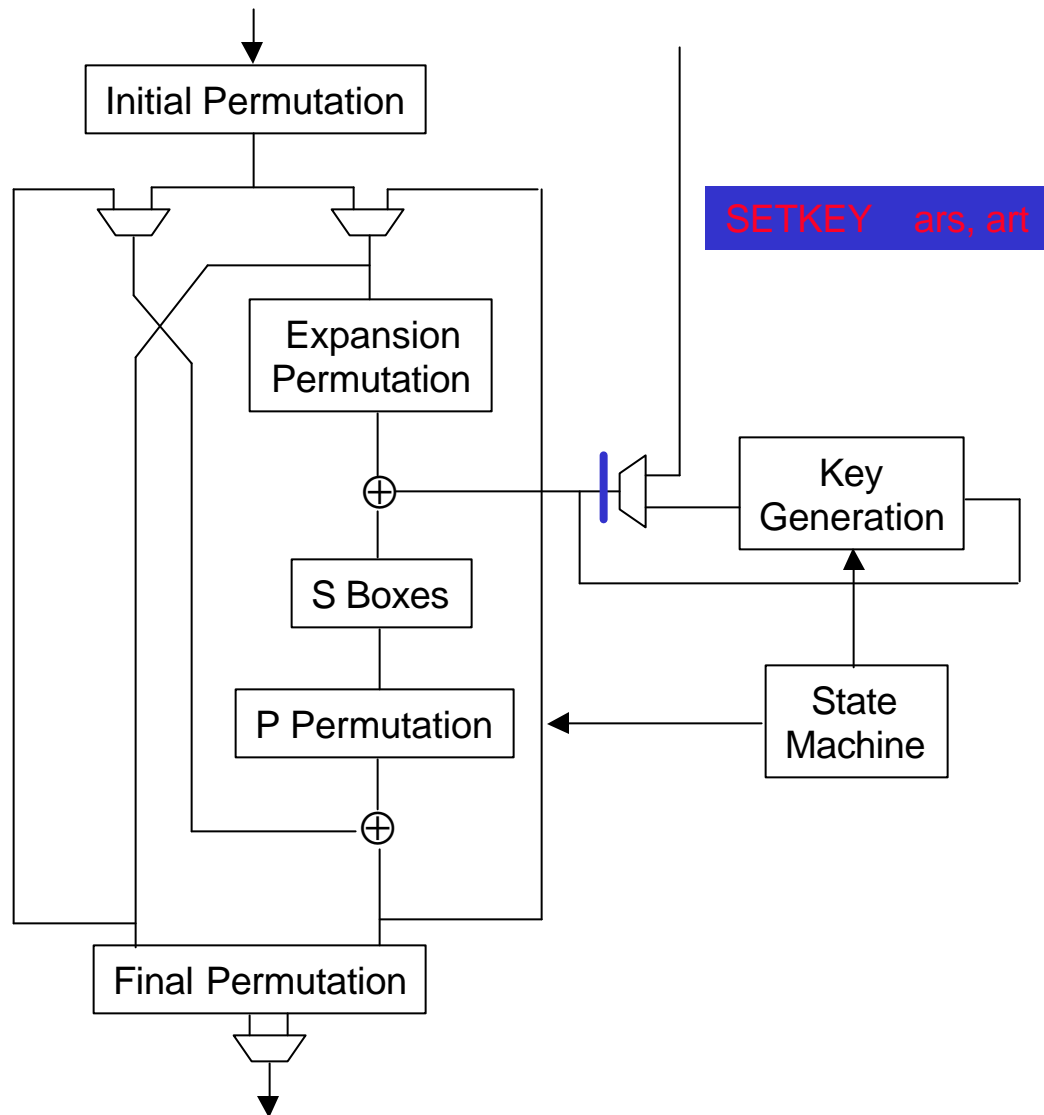
DES: Hardware Implementation



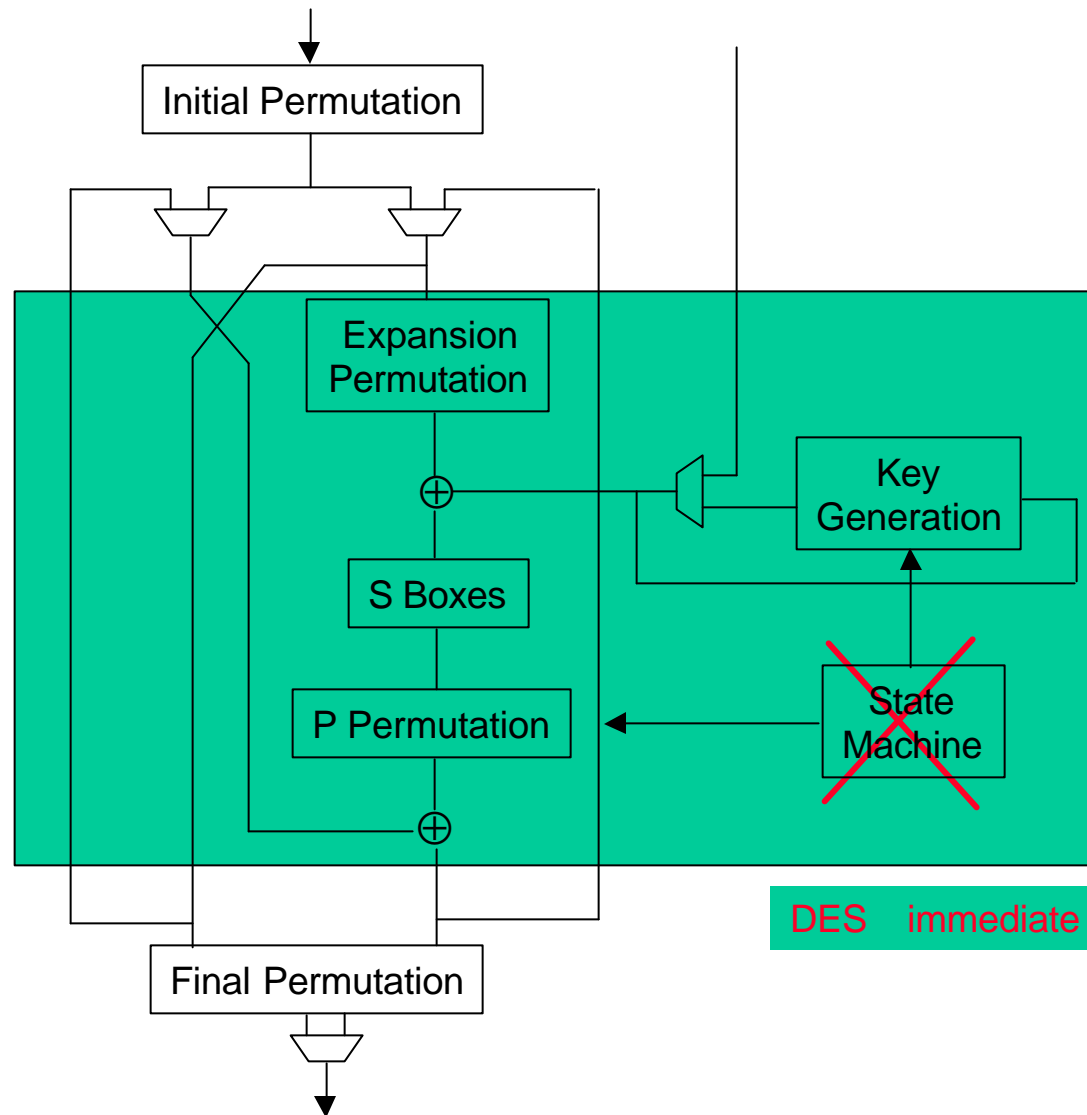
DES: SETDATA instruction



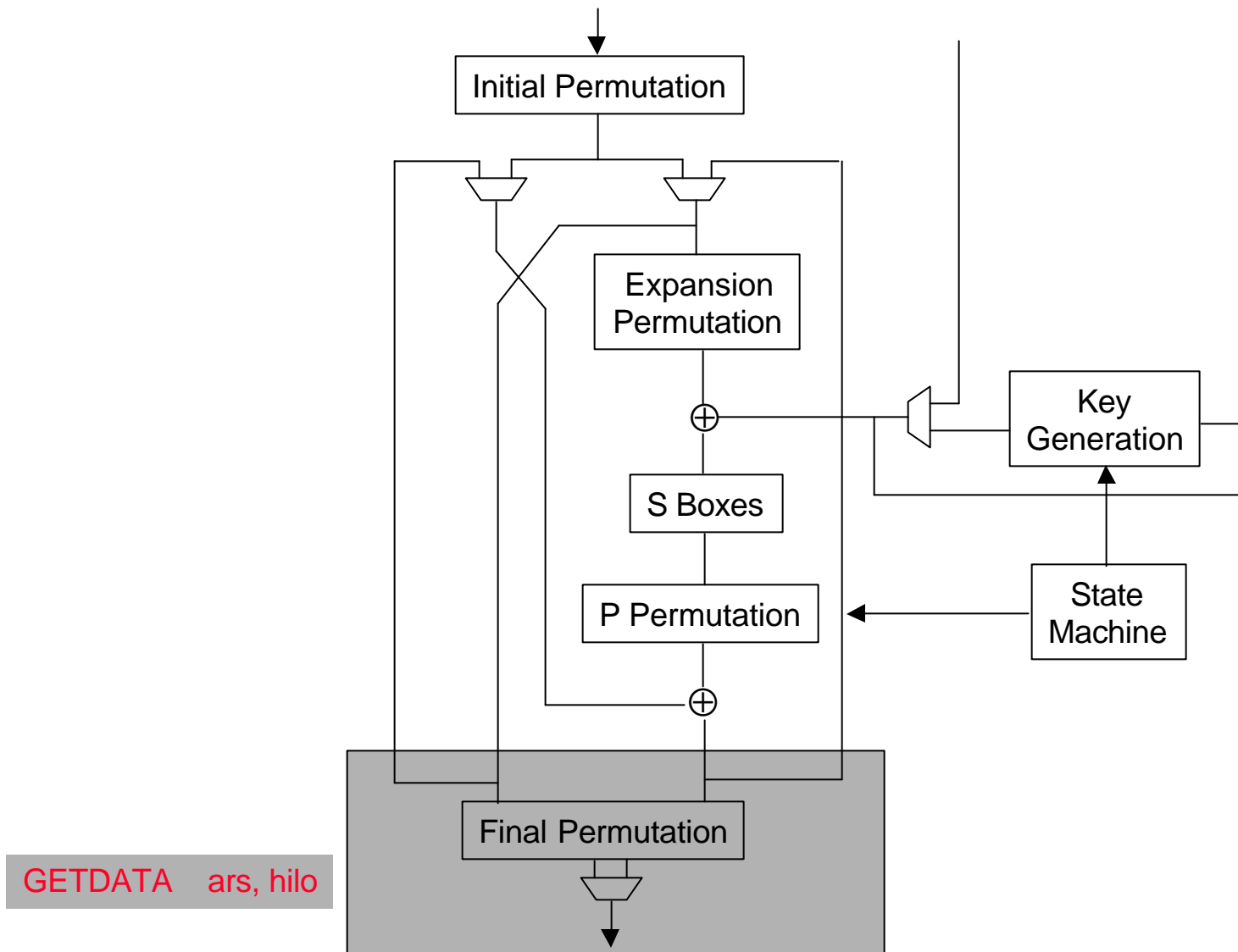
DES: SETKEY instruction



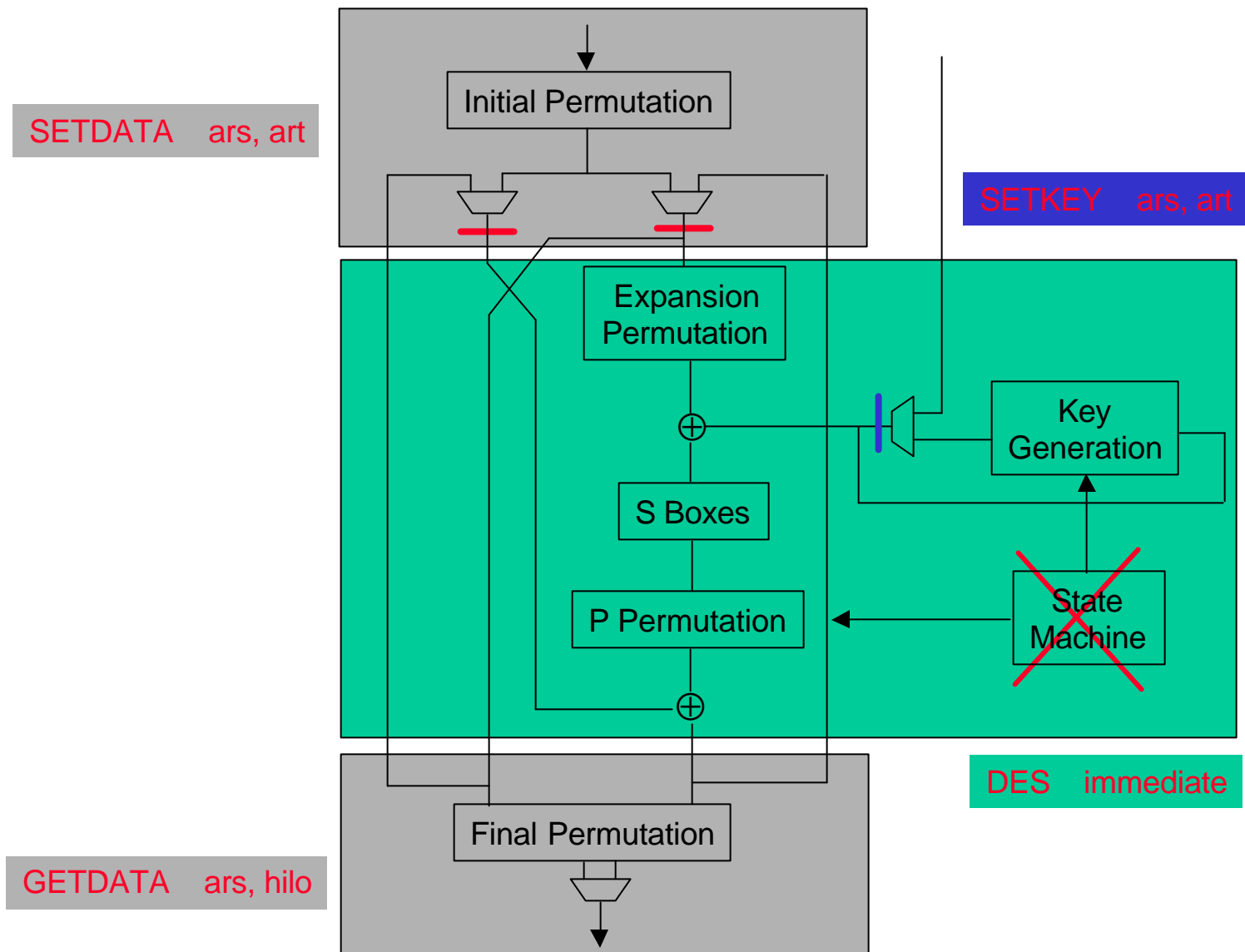
DES: *DES* instruction



DES: GETDATA instruction

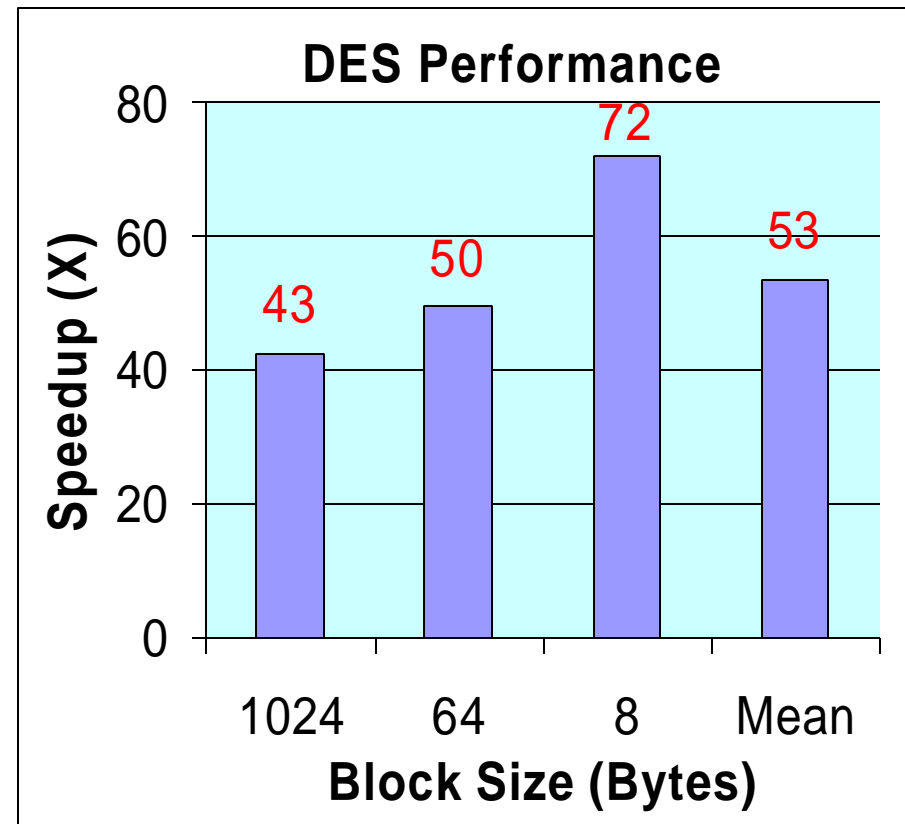


DES: Putting it together



DES: Summary

- ❖ Add 4 TIE instructions:
 - 80 lines of TIE description
 - No cycle time impact
 - ~1700 additional gates
 - Code-size reduced



❖ Configurable processors

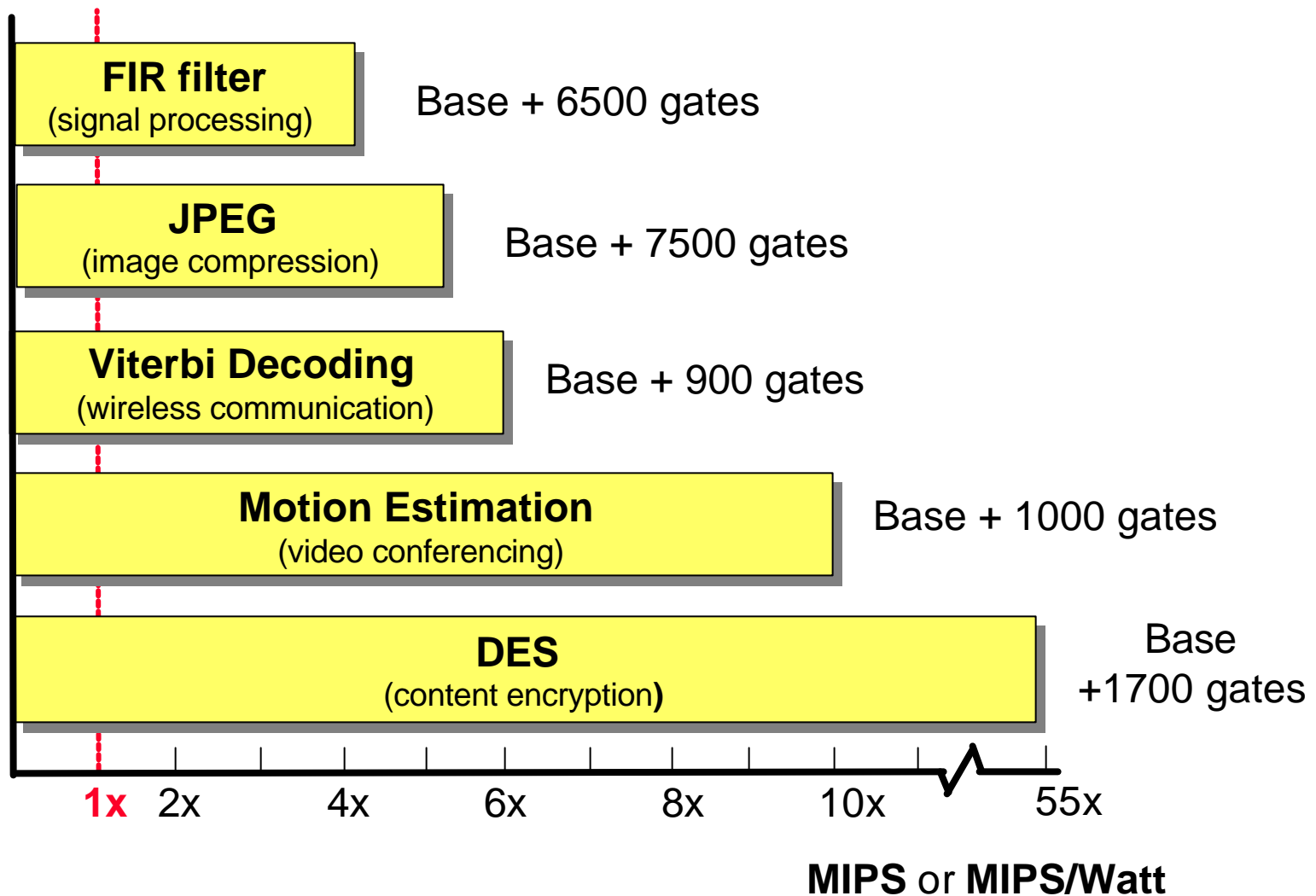
- Architecture
- Instruction extension
- Software support

❖ An Example

❖ Results

❖ Summary

Improvement over general purpose 32b RISC

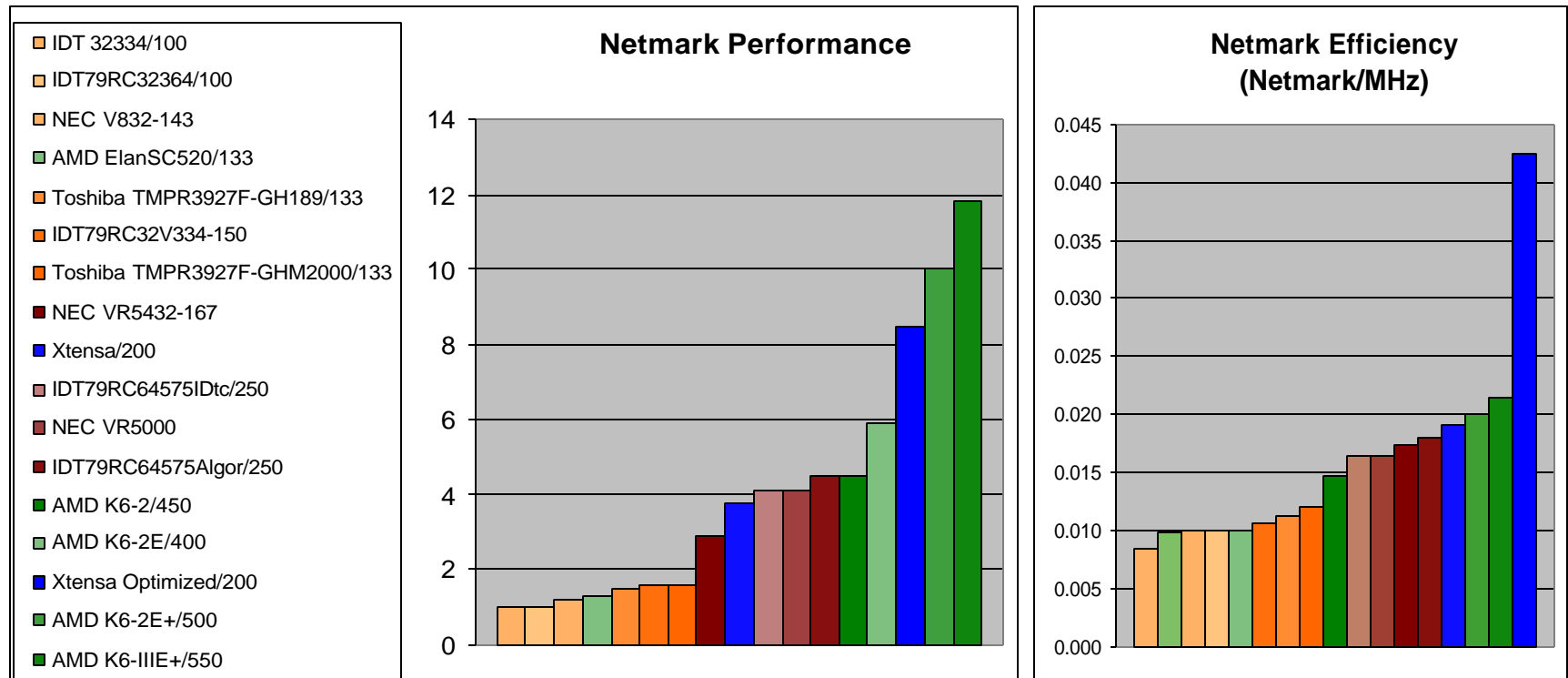


What is "EEMBC"?

- ❖ **EDN Embedded Microprocessor Benchmark Consortium**
- ❖ **Pronounced "Embassy"**
- ❖ **Non-profit consortium, funded by over 40 members**
 - Including: ARM, AMD, IBM, Intel, LSI Logic, MIPS, Motorola, National Semi, NEC, TI, Toshiba...Tensilica, and more...
- ❖ **Objective: Provide independently certified benchmark scores relevant to deeply embedded processor applications**
 - Independent laboratory recreates and certifies all benchmark results - no tricks
- ❖ **Five different benchmark suites:**
- ❖ **Each suite comprised of a range (five to sixteen) of benchmarks representative of that product category**
 - Example: Consumer: image compression, image filtering, color conversion

EEMBC Networking Benchmark

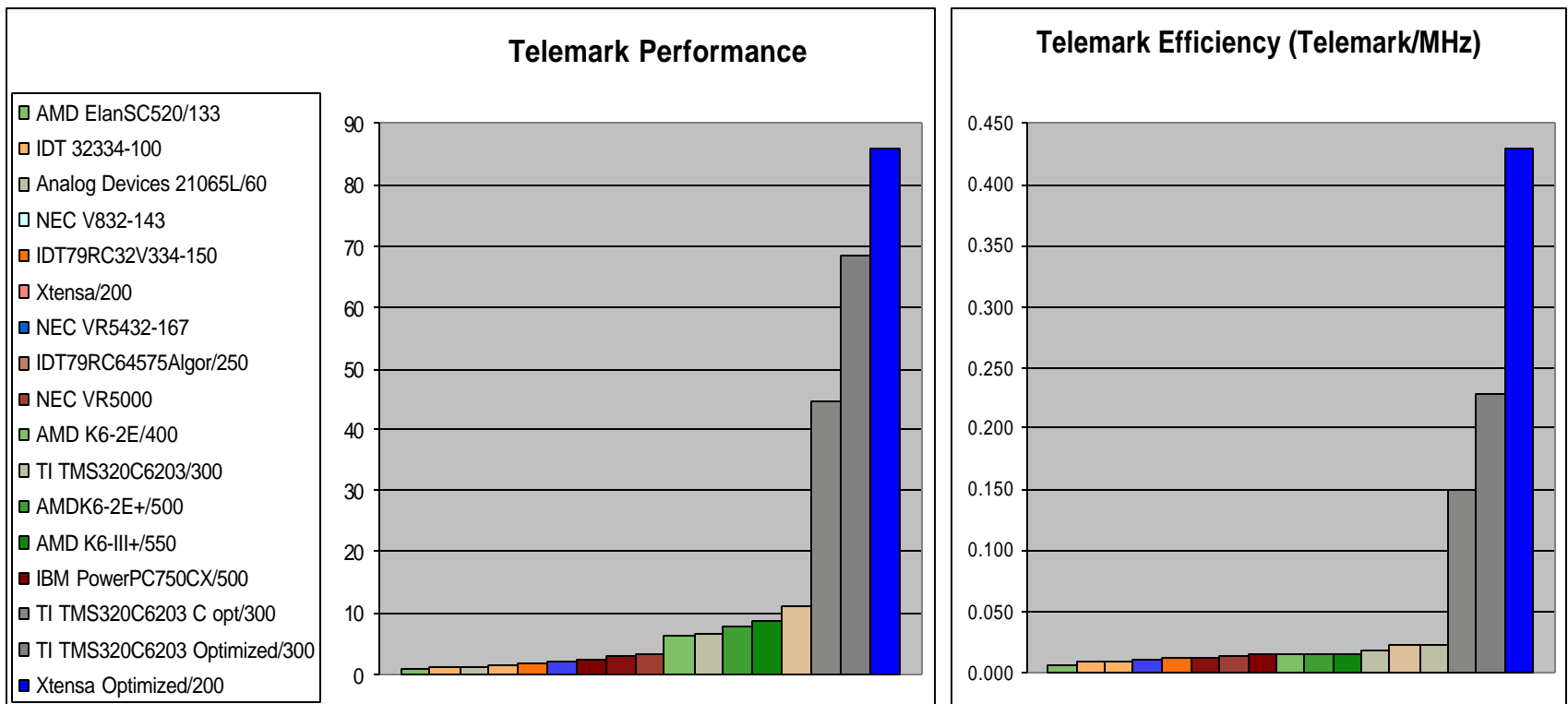
- ❖ Comparable in Netmark to high-end desktop CPUs
- ❖ 2x in Netmark/MHz
- ❖ 59K total gates at 200MHz



Colors: Blue-Xtensa, Green-Desktop x86s, Maroon-64b RISCs, Orange-32b RISCs

EEMBC Telecom Benchmark

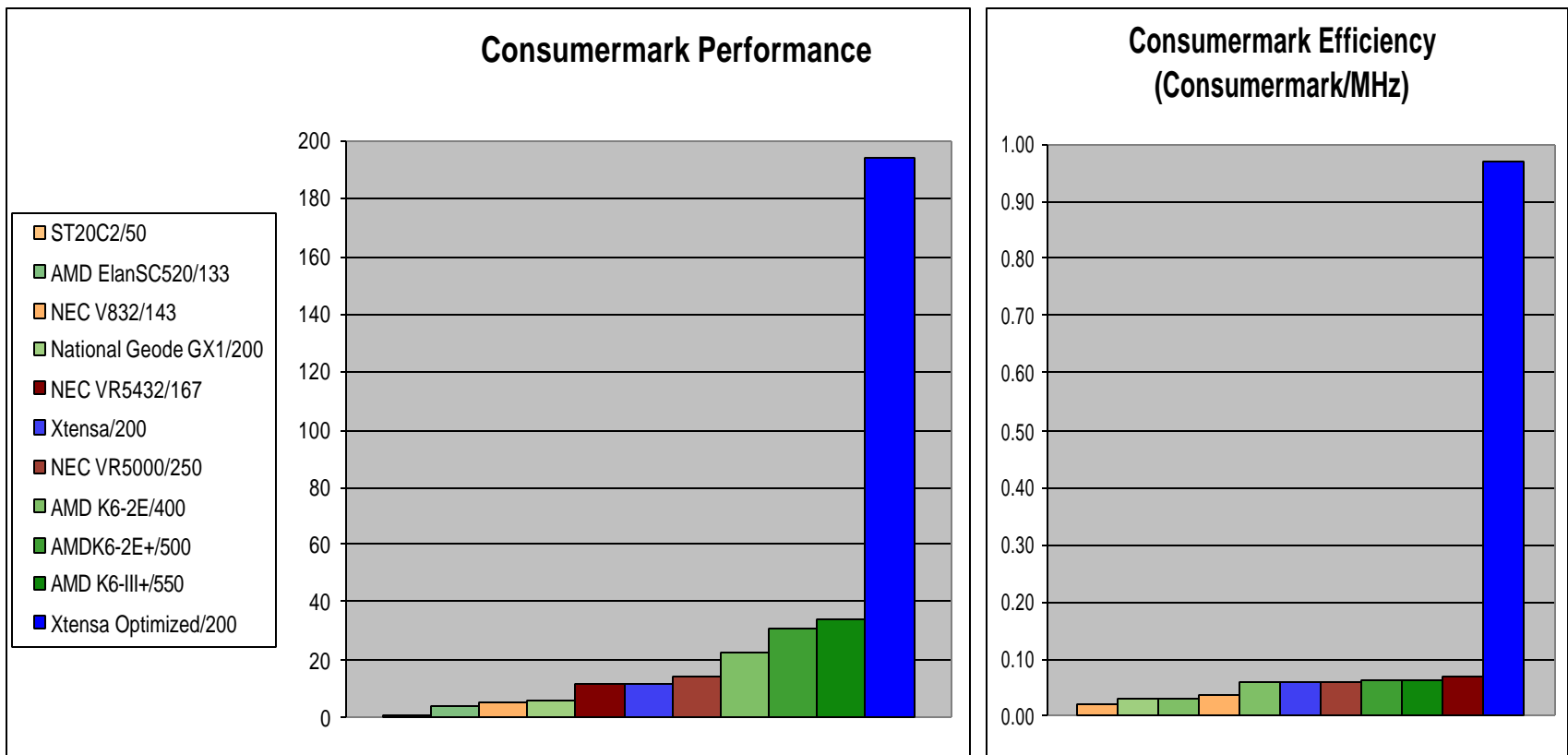
- ❖ Beats all processors, including hand-optimized TI C6x
- ❖ 180K total gates at 200MHz



Colors: Blue-Xtensa, Green-Desktop x86s, Maroon-64b RISCs, Orange-32b RISCs, Gray - DSPs

EEMBC Consumer Benchmark

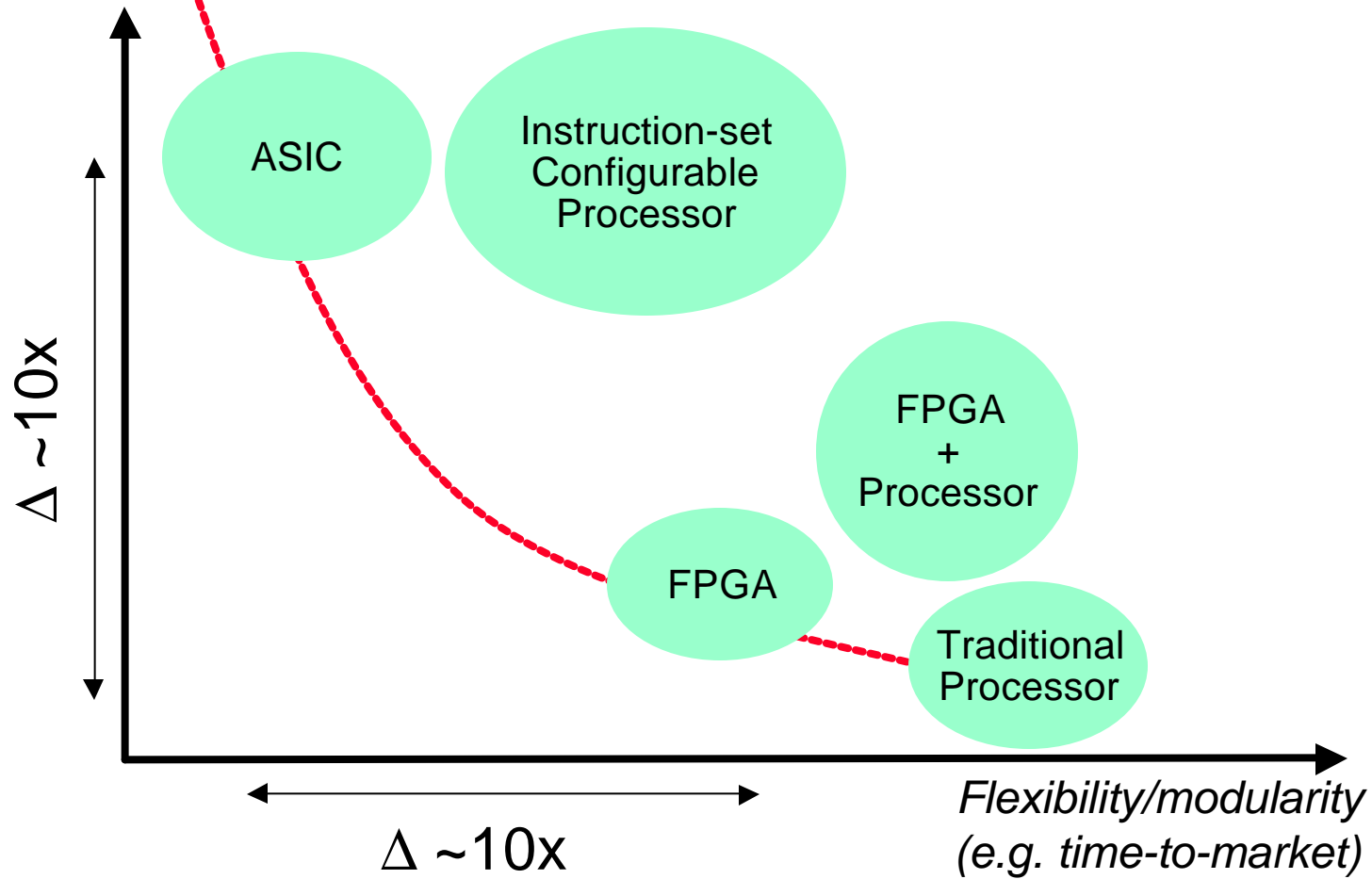
- ❖ 6x in Consumermark and 12x in Consumermark/MHz
- ❖ 127K total gates at 200MHz



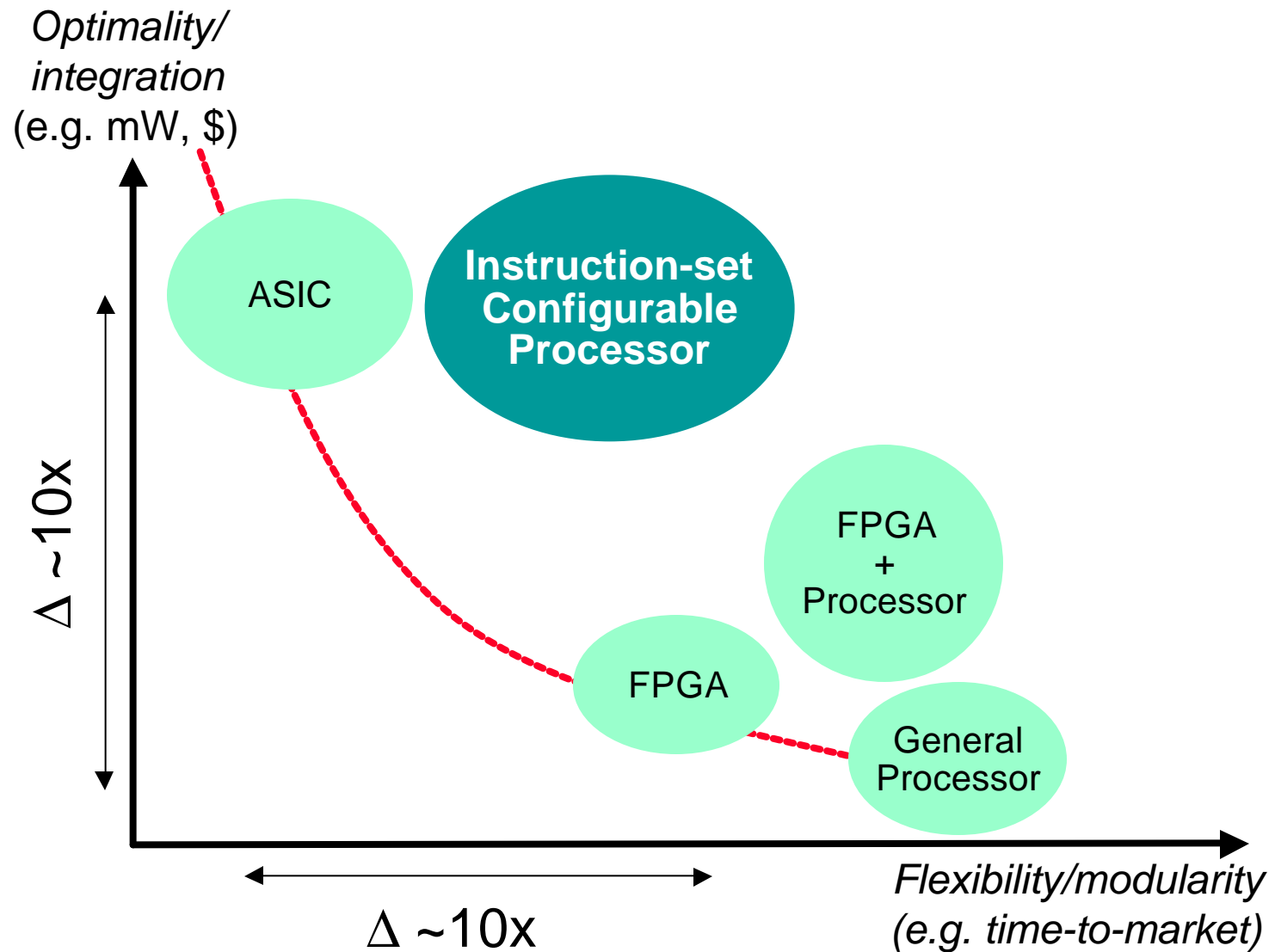
Colors: Blue-Xtensa, Green-Desktop x86s, Maroon-64b RISCs, Orange-32b RISCs

Summary

Optimality/
integration
(e.g. mW, \$)

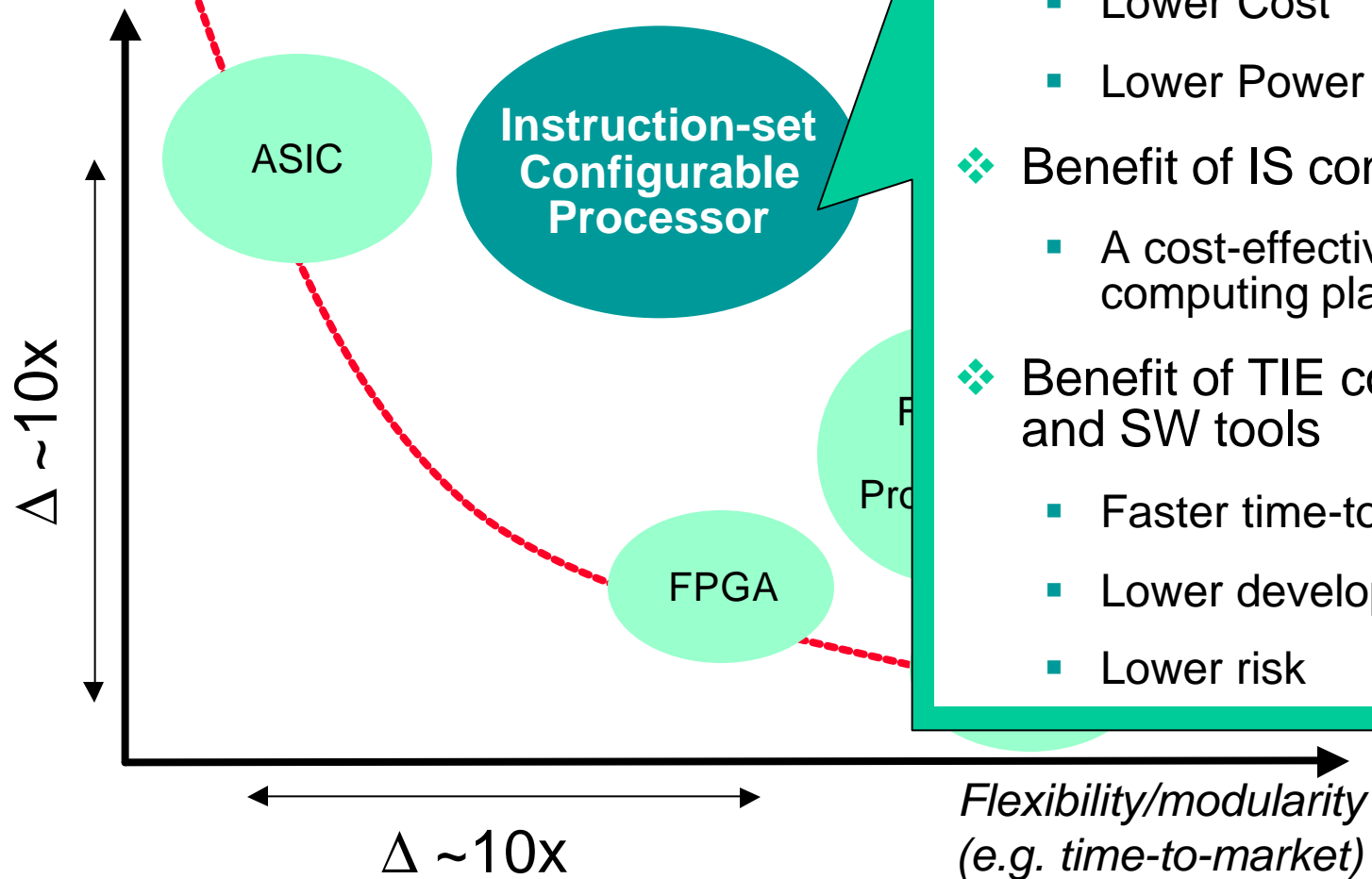


Summary



Summary

Optimality/
integration
(e.g. mW, \$)



Thank You!