Compiler Design for Digital Signal Processors

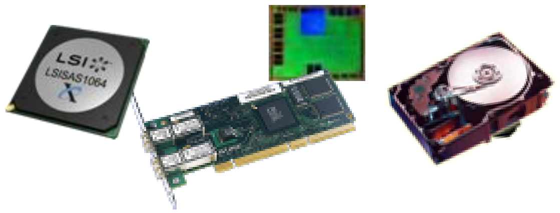
Erik Eckstein

03/25/2010

LSI Corporation

- LSI is a global leader in semiconductors and software solutions
- Solutions for storage and networking markets
- Founded 1981
- Headquarter: Milpitas, California
- Design centers around the world
  – Small design center in Vienna: development of DSP software tools

Market Focus

Storage		Networking
Systems	Semiconductors	
		
▪ Entry RAID Systems ▪ Mid-range RAID Systems ▪ Host RAID Adapters ▪ Host RAID SW ▪ Data Management SW	▪ HDD ICs: HDCs, SoCs, RCs, Pre-Amps ▪ SAS & SAS ROC Controllers ▪ SAS Expanders, Bridges ▪ SAN Fabric Products ▪ Custom Storage Products	▪ Custom Products – Enterprise Datacom ▪ Network Processors ▪ DSPs ▪ Framers/Mappers ▪ Link Layer Products ▪ Modems, USB, Firewire

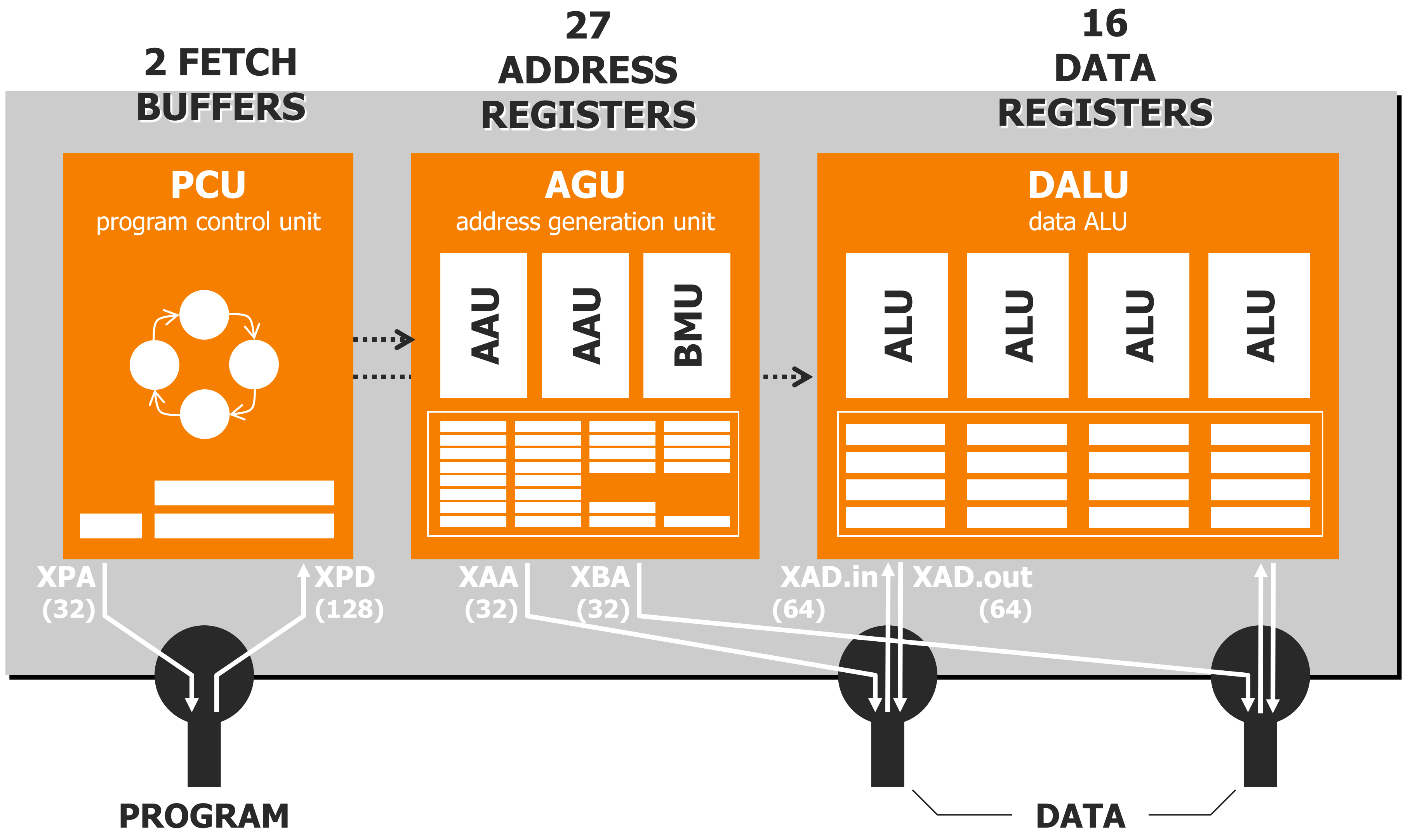
DSP Overview

- Digital Signal Processors
  - Used in mobile phones, wireless devices, infrastructure, etc.
  - Audio, video processing
  - High performance for signal processing algorithms
  - Low power consumption
- Example: StarPro2716
  - 16 *StarCore* SC3400e DSP cores @ 750 MHz
  - *8 ARM* cores for networking packet processing
  - Ethernet I/O support
  - 6MB on chip memory
  - 96 GMACs (96*10^9 multiply-accumulate instructions/sec)
  - Multichannel speech processing in voice gateways

DSP Characteristics

- RISC-like architecture
- Multiple parallel units
- VLIW (very long instruction word)
- Multiple memory buses
- 40-bit registers
- Complex addressing modes
- Hardware loops
- Fixed-point arithmetic

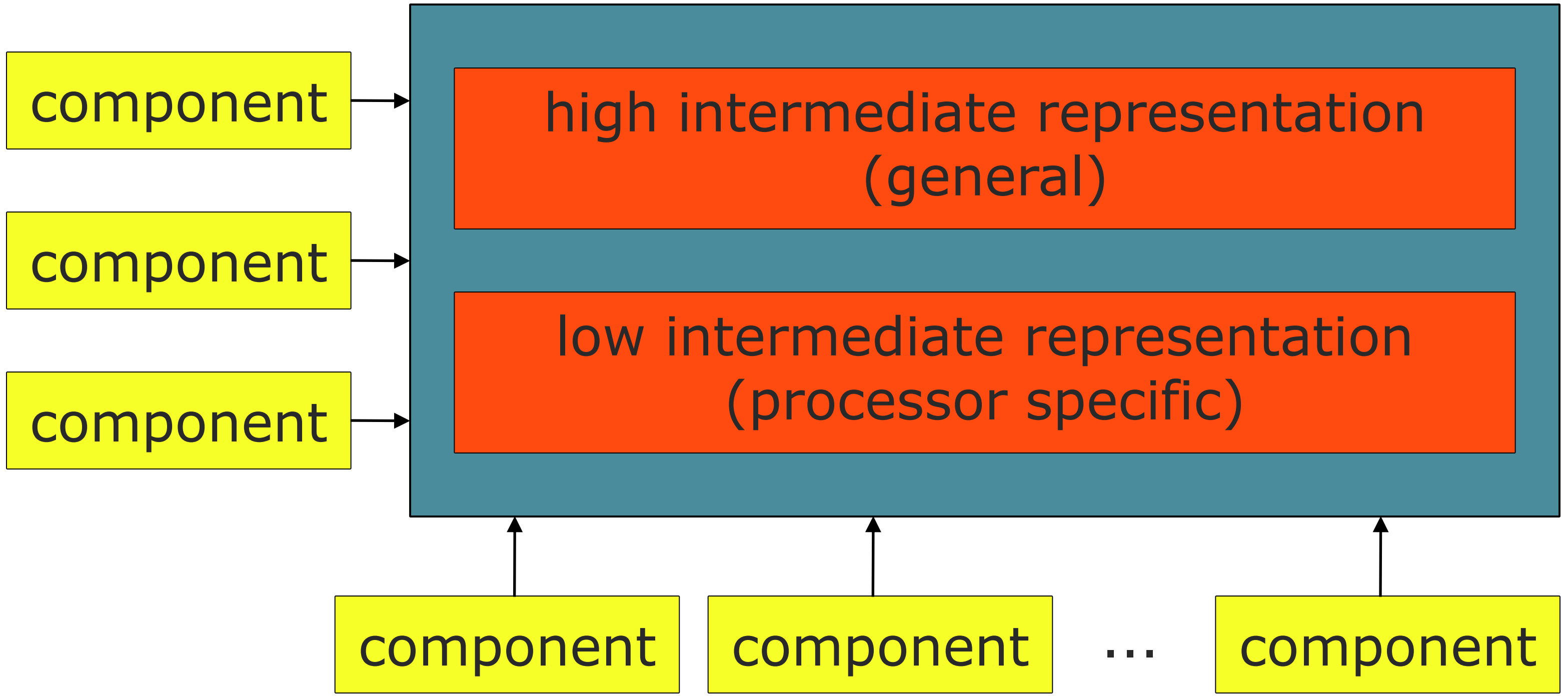
SC3400e Core



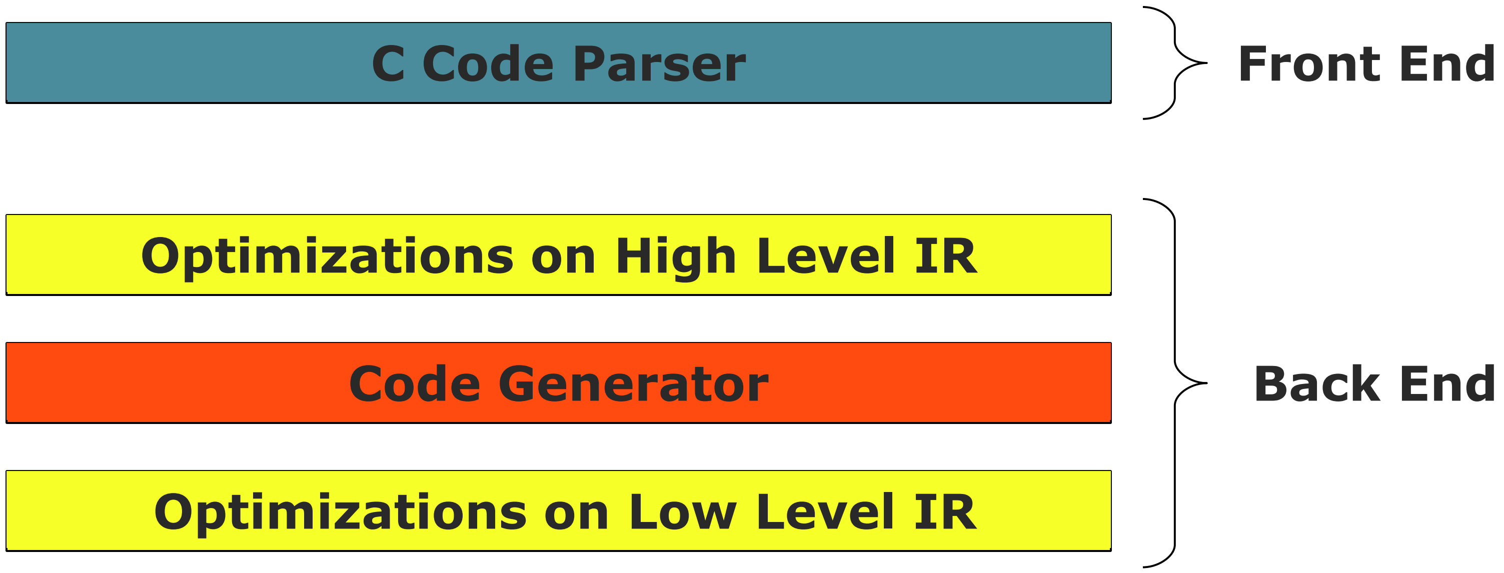
DSP-C Compiler Requirements

- Support for all DSP features
  - with language extensions (fractional data types)
  - with automatic recognition
- Optimizing for speed
  - time critical code
- Optimizing for code size
  - control code

Compiler Design

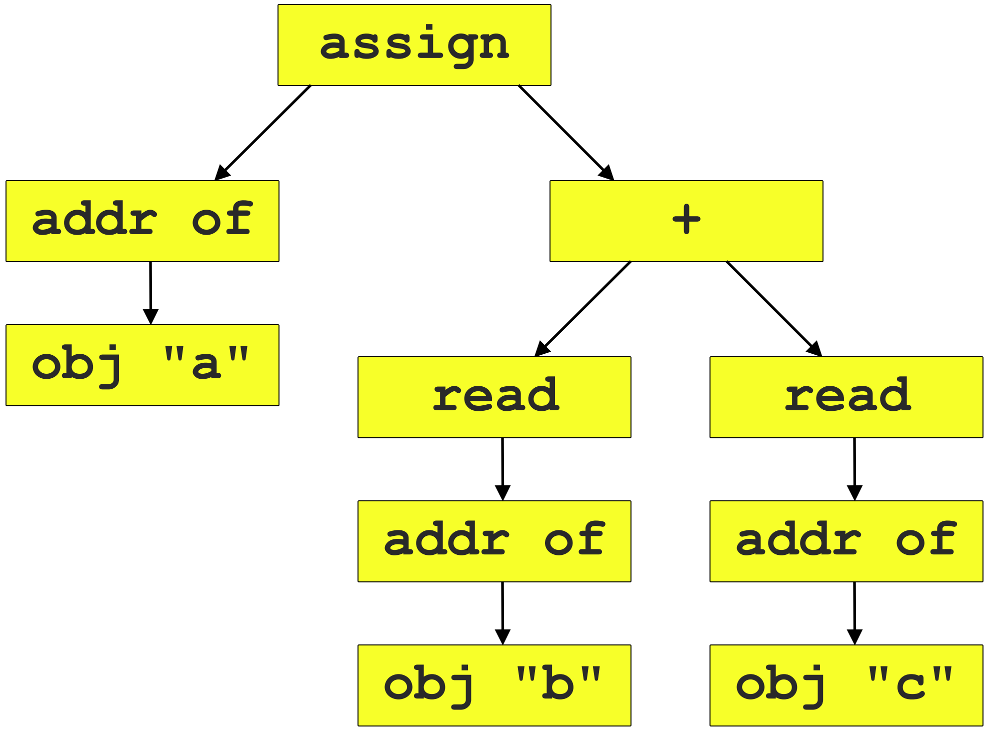

component
component
component
high intermediate representation (general)
low intermediate representation (processor specific)
component
component
…
component

Compilation Stages

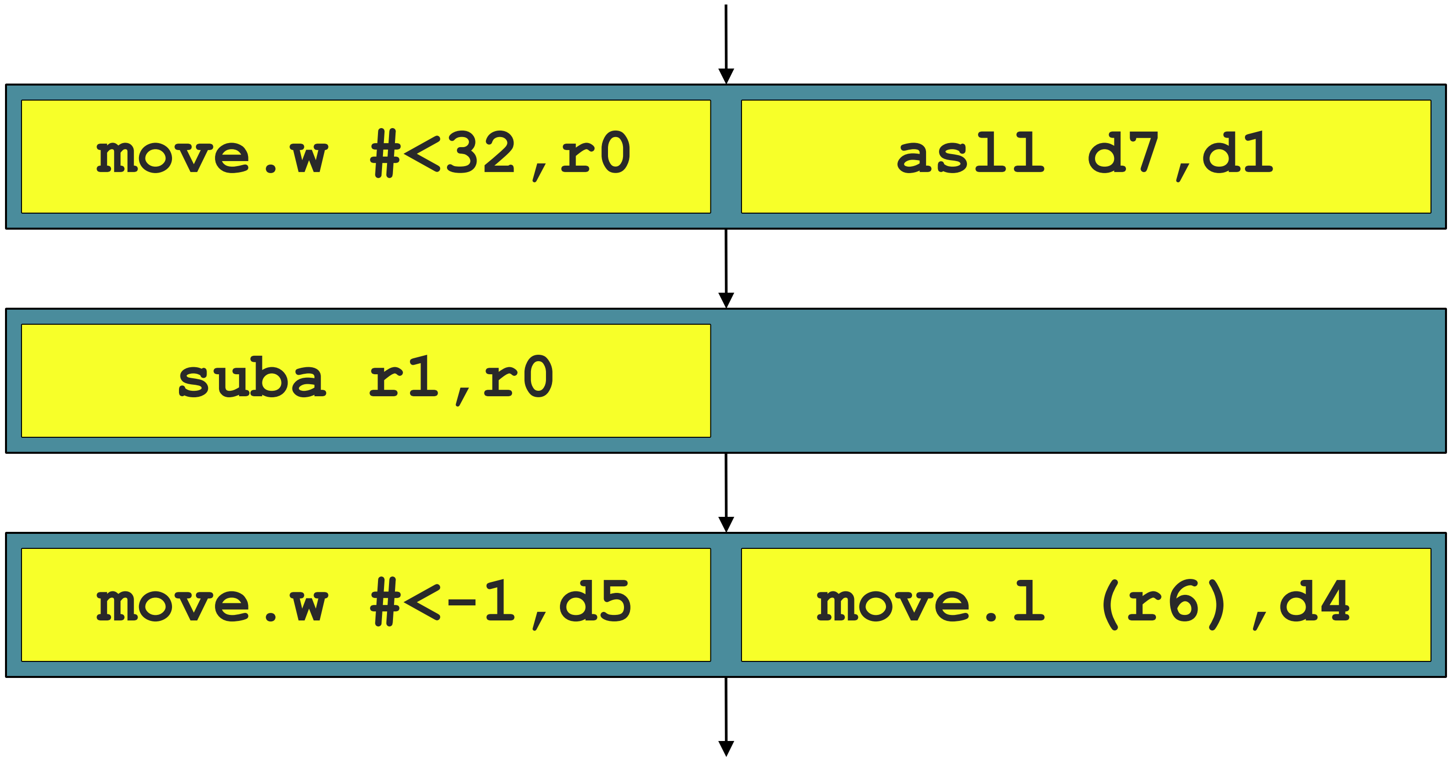


High Level IR Example

```
a = b + c;
```



Low Level IR Example



Optimizations are Important!

- Compiler contains approx. 50 different optimizations
  - – 70% high level optimizations
  - – 30% low level optimizations

- Most optimization are invoked several times
  - – up to 17 times!

Optimization Example: FIR Filter

```
long fir(short *x, short *y)
{
   long sum = 0;
   int i;
   for (i = 0; i < 128; i++) {
      sum += x[i] * y[i];
   }
   return sum;
}
```

Loop Strength Reduction

```
long fir(short *x, short *y)
{
    long sum = 0;
    int i;
    for (i = 0; i < 128; i++) {
        sum += *x * *y;
        x++;
        y++;
    }
    return sum;
}
```

Hardware Loop Support

```
long fir(short *x, short *y)
{
    long sum = 0;
    int i;
    loop 128 {
        sum += *x * *y;
        x++;
        y++;
    }
    return sum;
}
```

Instruction Selection

```
_fir
    doensh3   #128
    clr       sum
    loopstart3
      move.w    (x),tmp1
      move.w    (y),tmp2
      imac      tmp2,tmp1,sum
      adda      #4,x
      adda      #4,y
    loopend3
    rts
```

Register Allocation

```
_fir
    doensh3   #128
    clr       d0
    loopstart3
        move.w    (r0),d1
        move.w    (r1),d2
        imac      d2,d1,d0
        adda      #4,r0
        adda      #4,r1
    loopend3
    rts
```

5 cycles/iteration

Addressing Mode Optimization

```
_fir
    doensh3   #128
    sub       d0,d0,d0
    loopstart3
        move.w    (r0)+,d1
        move.w    (r1)+,d2
        imac      d2,d1,d0
    loopend3
    rts
```

3 cycles/iteration

Instruction Scheduling

```
_fir
    doensh3   #128
    sub       d0,d0,d0
    loopstart3
    [   move.w (r0)+,d1
        move.w (r1)+,d2 ]*)
        imac    d2,d1,d0
    loopend3
    rts
```

*) executed in parallel

2 cycles/iteration

Software Pipelining

```
_fir
   sub         d0,d0,d0
   doensh3   #127
   [ move.w (r0)+,d1  move.w (r1)+,d2 ]
   loopstart3
   [  move.w(r0)+,d1
      move.w(r1)+,d2
      imac d2,d1,d0 ]
   loopend3
   imac    d2,d1,d0
   rts
```

1 cycles/iteration

Loop Transformation

```
_fir
...
  loopstart3
  [    move.4w   (r0)+,d0:d1:d2:d3
       move.4w   (r1)+,d8:d9:d10:d11
       imac      d0,d8,d12
       imac      d1,d9,d13
       imac      d2,d10,d14
       imac      d3,d11,d1   ]
  loopend3
...
```

0.25 cycles/iteration

Questions?

LSI™