Very Low Power Pipelines using Significance Compression

Ramon Canal, Antonio Gonzalez and James E. Smith

Universitat Politecnica de Catalunya - Barcelona
University of Wisconsin - Madison
{rcanal, antonio}@ac.upc.es; jes@ece.wisc.edu

Motivation

- Size of the operands
  - Small values are largely used (that fit in a byte)
    - 00 00 00 00
    - FF FF FF FF
  - Large values may have bytes within that are just a sign extension of the previous one.
    - 07 7F 00 00
    - 01 00 00 00
  - Just the significant bytes need to be computed and stored; the rest are redundant

Significance Compression

- Conventional approach

<table>
<thead>
<tr>
<th>Style 1</th>
<th>Style 2</th>
<th>Style 3</th>
<th>Style 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="example.png" alt="Significant" /></td>
<td><img src="example.png" alt="Significant" /></td>
<td><img src="example.png" alt="Significant" /></td>
<td><img src="example.png" alt="Significant" /></td>
</tr>
</tbody>
</table>

- Sign extension bit

  Example: 0x00 - 0F 00 01 - 01 00 01 - 01 00 01
  - 1 extension bit per byte. byte0 is always computed as no sign extension needed

Talk Outline

- Significance compression
  - Extension bits
  - Basic pipeline
- Power savings
  - PC increment
  - Fetch
  - Register file
  - Functional units
  - Memory
  - Writeback

Conclusions

<table>
<thead>
<tr>
<th>Cases</th>
<th>% of when</th>
<th>Acc</th>
<th>Example (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxx</td>
<td>01.0</td>
<td>100</td>
<td>0000 0000</td>
</tr>
<tr>
<td>xxxx</td>
<td>13.0</td>
<td>75.0</td>
<td>0001 0000</td>
</tr>
<tr>
<td>xxxx</td>
<td>12.0</td>
<td>87.0</td>
<td>01 01 00 00</td>
</tr>
<tr>
<td>xxxx</td>
<td>11.0</td>
<td>94.0</td>
<td>01 01 00 00</td>
</tr>
<tr>
<td>xxxx</td>
<td>10.0</td>
<td>97.0</td>
<td>01 00 01 00</td>
</tr>
<tr>
<td>xxxx</td>
<td>9.0</td>
<td>99.0</td>
<td>01 01 00 00</td>
</tr>
<tr>
<td>xxxx</td>
<td>8.0</td>
<td>100</td>
<td>01 00 01 00</td>
</tr>
</tbody>
</table>

- Overall
  - 61.7% need 0 bits
  - 15.7% need 1 bit
  - 10.7% need 2 bits
  - 12.6% need 3 bits
Basic Pipeline

Five stage pipeline
- Fetch, Decode • Read, Execution, Memory, Writeback

Pipeline extended with extension bits
- Significant compression in all stages of the pipeline

Power Savings (i)

`PC increment`
- Compute 1 byte at a time
- Typically, carry does not propagate between bytes
  - Activity savings 73.3% average

Power Savings (ii)

`Fetch`
- Compress instructions (from 4 to 3 bytes)
  - Remove fields not used and compress function codes
  - Compression immediate that do not use all the space reserved
  - Most used instructions will use 3 bytes
  - 1 bit/4 bytes per instruction to distinguish among compressed and uncompressed instructions
  - Activity savings 18.2% average

Power Savings (iii)

`Register file (read)`
- Use significant compression when reading writing
  - Keep extension bits (3)
  - Activity savings 65.5% average

Power Savings (iv)

`ALU`
- Just compute “significant” bytes
- Generate extension bits for the result

- Activity savings 33.2% average

Power Savings (v)

`Memory (D-cache data)`
- Just send/receive “significant” bytes
  - Keep extension bits in the cache

- Activity savings 30.1% average
Power Savings (vi)

- Register file (write)
  - Just write significant bytes
  - Update extension bits
- Activity savings 42.1% average

Power Savings (vii)

- Datapath latches
  - Keep flowing significant bytes
  - Keep extension bits
- Activity savings 42.2% average

Power Savings (viii)

Summary power savings

<table>
<thead>
<tr>
<th>Datapath Width</th>
<th>PC incr</th>
<th>Fetch</th>
<th>RF read</th>
<th>ALU</th>
<th>D cache data</th>
<th>RF write</th>
<th>Latches</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>75.3%</td>
<td>18.2%</td>
<td>46.5%</td>
<td>22.1%</td>
<td>23.4%</td>
<td>30.3%</td>
<td>34.9%</td>
</tr>
<tr>
<td>16 bit</td>
<td>46.7%</td>
<td>18.2%</td>
<td>35.9%</td>
<td>22.1%</td>
<td>23.4%</td>
<td>30.3%</td>
<td>34.9%</td>
</tr>
</tbody>
</table>

Low Power Pipelines

Serial
  - Minimal activity

Semi-parallel
  - Throughput balanced

Fully-parallel
  - Complex control
  - Aggressive open and gating

Similar activity counts, different performance levels

Low Power Pipelines

Byte-serial implementation

Tag compare

Byte semi-parallel implementation

Tag compare
Summary and Conclusions

Significant bytes determine minimal activity
1. For instructions, addresses and data values
2. This level is 30-40% lower than a conventional 32-bit architecture

Proposed pipeline implementations
1. Several designs proposed
2. Trading off power consumption and performance
   - Similar activity reduction levels
   - Minimal activity achieved by the parallel organization
   - Lower activity
   - Similar data consumption (forwarded and idle blocks)
   - Maximum performance achieved by the parallel organization