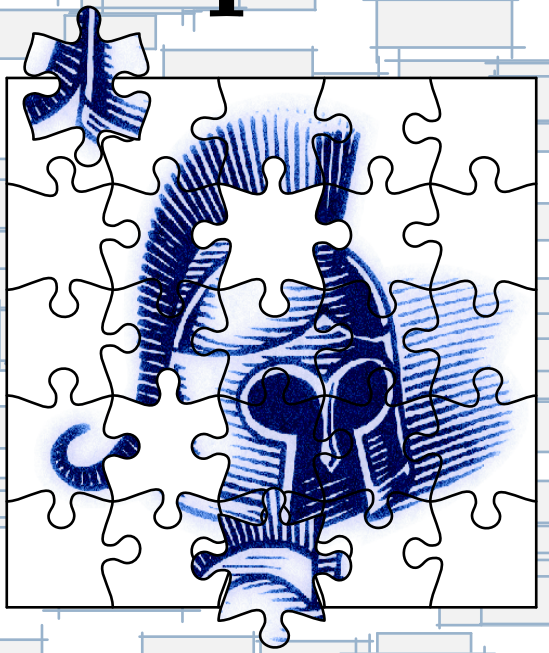
GCC Backend for the Ulysse processor



Michel Ganguin
supervised by:
Dr. Michel Schinz
Pierre-André Mudry
and
Prof. Auke Ijspeert

Master Thesis
Computer Science
Fall Semester 2007-2008

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

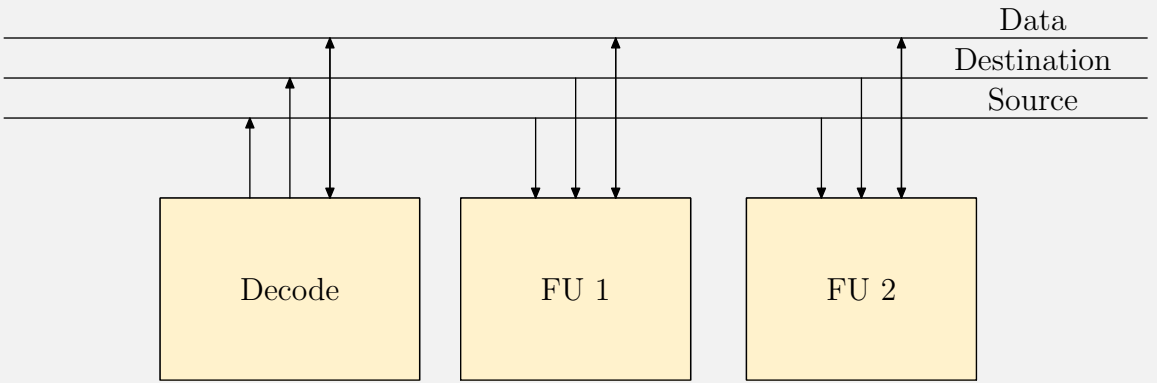
Presentation overview

Introduction

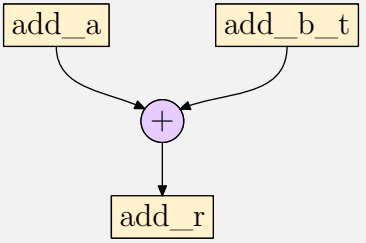
- A C compiler for the Ulysse processor.
- Use of an existing compiler, GCC.
- GCC proposes a framework to port it.
- Simpler than rewriting a compiler from scratch?

MOVE Arch

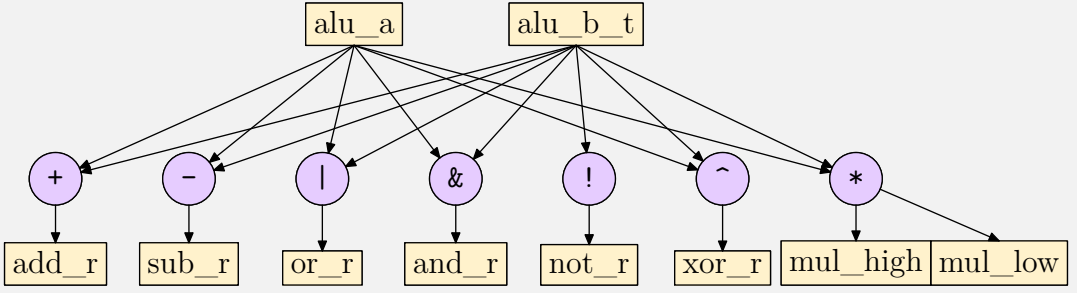
- Transport Triggered Architecture.
- Only one instruction: *move*.
- Functional units on processor's bus.



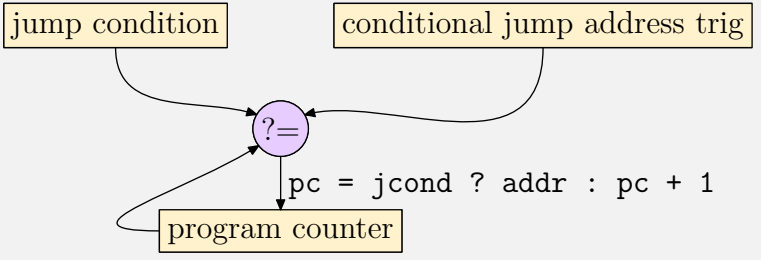
Adder example



ALU example



Conditional jump



MOVE pros and cons

Software view:

- Instruction scheduling flexibility
- Less register file transfers
- Difficult to program (Compiler!?)
- Program size might be bigger

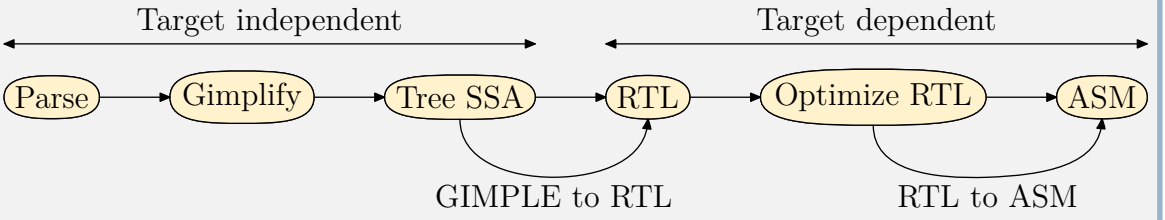
Hardware view:

- Simple decode logic
- Scalability: new FUs, number of busses (VLIW)

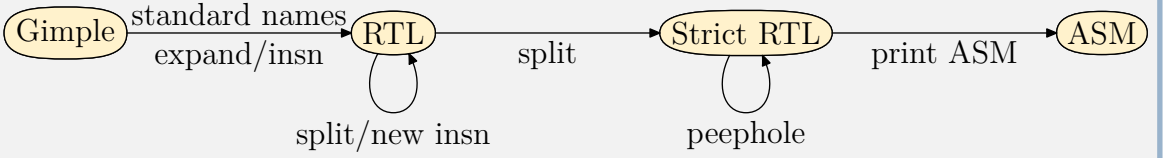
Performance view:

- Benchmarks are often written in C :)
- Seems not to be to most efficient!

GCC Overview



- `move.h` Target macros
- `move.md` Machine description



Target Macros

```
//...
#define WORDS_BIG_ENDIAN 0
#define BITS_PER_UNIT 32
#define BITS_PER_WORD 32
//...
#define STACK_GROWS_DOWNWARD
#define FRAME_GROWS_DOWNWARD 1
#define STACK_POINTER_REGNUM 1
#define FRAME_POINTER_REGNUM 2
//...
#define ASM_OUTPUT_LABELREF(stream, name) \
  fprintf(stream,"%s",name);
//...
#define Pmode QImode
#define FUNCTION_MODE QImode
```

Machine description

```
(define_insn "andqi3"
 [(set (match_operand:SI 0 "and_result_register_operand"
                           "=Srandr,Srandr")
       (and:SI (match_operand:QI 1 "alu_a_register_operand"
                                   "Sralua,Sralua")
               (match_operand:QI 2
                     "readable_register_or_immediate_operand"
                     "Rra,i")))
  (clobber (reg:QI SUB_R_REGNUM))
  (clobber (reg:QI OR_R_REGNUM))
  (clobber (reg:QI ADD_R_REGNUM))
  (clobber (reg:QI NOT_R_REGNUM))
  (clobber (reg:QI XOR_R_REGNUM))
  (clobber (reg:QI MUL_LOW_REGNUM))
  (clobber (reg:QI MUL_HIGH_REGNUM))
 ]
 ""
 "@
  move \\talu_b_t, %2;\t//and
  move \\talu_b_t, %2;\t//immediate and")
```

Examples

```
int a, b, c;
a = b + c;
```

```
(set (mem:QI (symbol_ref:QI ("a")))
     (plus:QI (mem:QI (symbol_ref:QI ("b")))
              (mem:QI (symbol_ref:QI ("c")))
     )
)
```


Examples – cont.

```
int a, b, c;
a = b + c;
```

```
(set (reg:QI mem)
     (mem:QI (symbol_ref:QI ("b"))))
(set (reg:QI alu_a)
     (reg:QI mem))
(set (reg:QI mem)
     (mem:QI (symbol_ref:QI ("c"))))
(set (reg:QI add_r)
     (plus:QI (reg:QI alu_a)
              (reg:QI mem)))
(set (reg:QI mem)
     (reg:QI add_r))
(set (mem:QI (symbol_ref:QI ("a")))
     (reg:QI mem)
```

Examples – cont.

```
int a, b, c;
a = b + c;
```

```
move load_abs_addr, #b;
move alu_a, mem;
move load_abs_addr, #c;
move alu_b_t, mem;
move mem, add_r;
move store_abs_addr, #a;
```


Examples – cont.

```
int a, b, c;
a = b + c;
```

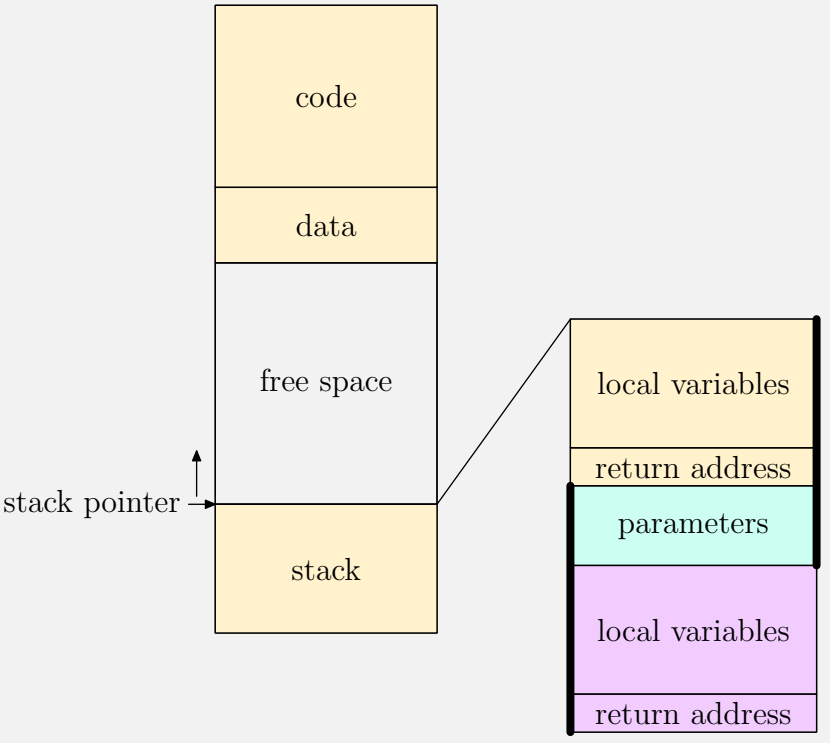
```
move load_abs_addr, #b;
move alu_a, mem;
move load_abs_addr, #c;
move alu_b_t, mem;
move mem, add_r;
move store_abs_addr, #a;
```

```
move load_abs_addr, #b;
move r0, mem;
move alu_a, r0;
move load_abs_addr, #c;
move r1, mem;
move alu_b_t, r1;
move r0, add_r;
move mem, r0;
move store_abs_addr, #a;
```

`asm` statment

```
void func(int a, int b){
  asm("move %0, %1":"=Rwa"(a):"Rra"(b):"r"(b));
}
```

Memory usage



Particularities

- readonly and writeonly registers
- trigger registers
- Almost one register class per register
- load, store and add split/macros

Further work

- Handle the case of spilling readonly and writeonly registers
- Make further tests
- Find a flexible way to add new FUs
- Latency aware instruction scheduling and other optimizations
- Low-level runtime libraries
- Standard C runtime libraries
- A linker

Conclusion

- RTL is much simpler than C
- Ulysse assembler code is comparable to RISC with heavy register constraints
- We used some ”macros” to make it work
- All target independent optimizations of GCC can be used
- With runtime libraries, all possible C code should be compilable

Collection of Howlers

- For GCC a ”byte” is the smallest addressable data, i.e. 32bits in our case.
- ”Reload is the GCC equivalent of Satan” according to http://gcc.gnu.org/wiki/reload.
- Some global variables have sometimes exactly the opposite value that they should have when used in function where they aren't aimed for.
- Some constants has to be declared twice because of "the chicken or the egg" dilemma.

References

- Stallman, R. M. and the GCC Developer Community (2007). *GNU Compiler Collection Internals. For GCC version 4.3.0.* Free Software Foundation.
- Nilsson, H.-P. (2000). *Porting GCC for Dunces.* Axis Communication.
- Mudry, P.-A. (2004). *Système multiprocesseurs on-chip: Étude et réalisation à base de processeurs utilisant une architecture move.* Master's thesis, EPFL.
- Deshpande, S. and Khedker, U. P. (2007). *Incremental Machine Description for GCC.* Indian Institute of Technology, Bombay.
- Centre for Formal Design and Verification of Software and Dept. of Computer Science & Engineering, IIT Bombay (2007). *Workshop on GCC Internals.*