hack.ast

```
Require Export hack.ast.reg.
Require Export hack.ast.instruction.
Require Import hack.result.
```

The abstract syntax.

This module exposes the abstract syntax of the hack assembly language program. An assembly language program consists of a list of labeled block of hack machine instructions. Instructions and registers are captured in the modules below.

Blocks

Blocks are merely list of instructions. We also have a labeled variant of blocks and these types are captured in the modules below. The map and resolve functions from instructions carry over to these.

```
Module block.

  Definition t v := list (instruction.t v).

  Definition wordSize {v} : t v -> N := List.fold_right (fun _ x => x + 1)%N 0%N.

  Definition map {u v} (f : u -> v) : t u -> t v := List.map (instruction.map f).

  Definition resolve {u v}(f : u -> option v) : t u -> result.t (t v) u :=
    result.collect \o List.map (instruction.resolve f).

  Module labeled.

    Definition t v := (v * block.t v)%type.

    Definition map {u v}(f : u -> v) (blk : t u) : t v :=
      let (xu, iu) := blk in
      (f xu, block.map f iu).

    Definition resolve {u v}(f : u -> option v)(blk : t u) : result.t (t v) u :=
      map2 pair (result.option.app f blk.1) (block.resolve f blk.2).

  End labeled.
End block.
```

Hack assembly program

An assembly program in hack is an initial unlabeled block followed by a

```
Module program.

  Record t {v} := Program { preamble : block.t v;
                           body : list (block.labeled.t v)
                  }.

  Arguments t : clear implicits.
  Arguments Program {v}.

  Definition map {u v}(f : u -> v) (pgm : t u) : t v :=
    {|
      preamble := block.map f (preamble pgm);
      body := List.map (block.labeled.map f) (body pgm)
    |}.

  Definition resolve {u v}(f : u -> option v)(pgm : t u) : t v + list u :=
    let prble := block.resolve f (preamble pgm) in
    let bdy := result.collect (List.map (block.labeled.resolve f) (body pgm)) in
    result.map2 Program prble bdy.

  Definition labelDef v := list (v * N).
  Definition labelDefCombine {v}(acc : labelDef v * N) (vblk : block.labeled.t v) :
labelDef v * N :=
    let (vs, offset) := acc in
    ((vblk.1 , offset) :: vs, offset + block.wordSize (vblk.2))%list%N.

  Definition address {v}(prog : t v) : labelDef v :=
    let fstLabel := block.wordSize (preamble prog) in
    (List.fold_left labelDefCombine (body prog) ([]%list,fstLabel)).1.

End program.
```

Combinators for some common instructions.

This module exposes convenient functions to build common instructions that are needed in practice. The main use case is to generate hack assembly language programatically in other applications (for example a compiler that targets hack architecture). Such applications can bypass the parsing and lexing phase and directly generate values in the instruction type.

```
Module mnemonics.

  Section Instructions.
    Context {v : Type}.

    Definition jumpOf (j : jump) : instruction.t v :=
    match j with
    | JMP => instruction.C [] (constant Zero) (Some JMP)
    | _ => instruction.C [] (uapply ID reg.D) (Some j)
    end.

    Definition jgt := jumpOf JGT.
    Definition jeq := jumpOf JEQ.
    Definition jge := jumpOf JGE.
    Definition jlt := jumpOf JLT.
    Definition jne := jumpOf JNE.
    Definition jle := jumpOf JLE.
    Definition jmp := jumpOf JMP.
```

```
      Definition add : reg.AOrM -> output := bapply Add.
      Definition sub : reg.AOrM -> output := bapply Sub.
      Definition subfrom : reg.AOrM -> output := bapply SubFrom.
      Definition band : reg.AOrM -> output := bapply BAnd.
      Definition bor : reg.AOrM -> output := bapply BOr.
      Definition uminus : reg.t -> output := uapply UMinus.
      Definition bneg : reg.t -> output := uapply BNeg.
      Definition succ : reg.t -> output := uapply Succ.
      Definition pred : reg.t -> output := uapply Pred.

      Definition assign (r : reg.t)(o : output) : instruction.t v :=
        instruction.C [r] o None.

    End Instructions.

  End mnemonics.
```

Index

This page has been generated by coqdoc