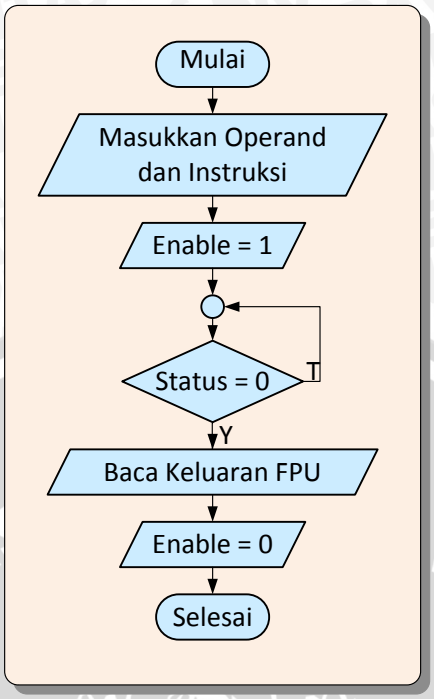
BAB IV
PERANCANGAN

Bab ini membahas tentang perancangan FPU yang dapat menghitung bilangan *floating point*32 bit sesuai standard IEEE 754 yang diaplikasikan guna membantu proses perhitungan bilangan floating point dalam FPGA. Perancangan FPU ini meliputi perancanganSistem FPU, Pembangunan *Hardware* FPU, Algoritma FPU, dan Perancangan komponen FPU.

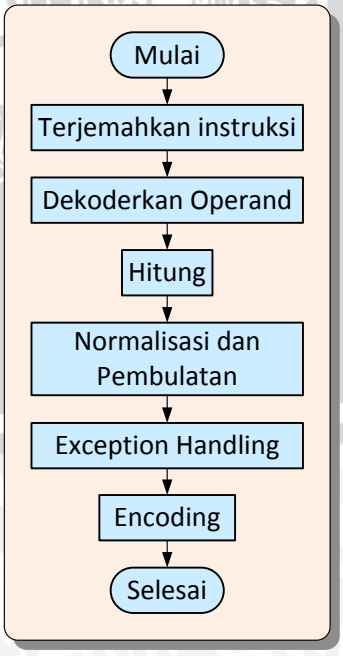
4.1 Sistem FPU

FPU yang dirancang pada penelitian ini adalah FPU yang bekerja dengan cara mengeksekusi perintah dengan operand yang sudah ditentukan ketika sinyal *enable* diaktifkan (logika 1), dan akan memberi sinyal *busy* (logika 1) ketika sedang menghitung dan menonaktifkan sinyal *busy* ketika proses perhitungan telah selesai. Aturan-aturan tersebut diciptakan untuk mempermudah seseorang yang akan memanfaatkan sistem FPU ini kedalam sistem lain. Sistem FPU ini bekerja secara pasif, artinya FPU akan melakukan perhitungan jika sinyal *enable* diaktifkan, ketika sinyal *enable* non-aktif maka FPU tidak akan melakukan perhitungan (*standby*). Alur pengerjaan FPU dibagi menjadi enam, yaitu pemberian *operand*, pengaktifan sinyal *enable*, menunggu sinyal *busy* sampai tepi turun, pembacaan hasil perhitungan FPU, dan penon-aktifan sinyal *enable*. Secara umum diagam alir pengerjaaan FPU ditunjukkan pada Gambar 4.1



Gambar 4.1 Diagram alir pengerjaan FPU

Dan di dalam FPU, FPU baru akan melakukan proses perhitungan ketika sinyal *enable* tepi naik. Alur proses FPU dibagi menjadi 5 bagian, yaitu menerjemahkan instruksi, mendekoderkan operand, menghitung, menormalisasi hasil perhitungan dan melakukan pembulatan, *exception handling* (penanganan perhitungan khusus) dan kemudian *encoding*. Secara umum alur proses FPU ditunjukkan pada gambar 4.2



Gambar 4.2 Diagram alir proses perhitungan pada FPU

Namun bila dijabarkan lebih mendetil, alur proses perhitungan FPU dibagi menjadi tiga tergantung dari jenis perhitungan yang akan diproses oleh FPU. Jika proses perhitungan adalah penjumlahan atau pengurangan, kedua eksponen dibandingkan, eksponen hasil perhitungan adalah eksponen yang terbesar, mantissa bilangan *floating point* dengan eksponen yang lebih kecil digeser kekanan sebanyak selisih kedua eksponen. Kedua mantissa kemudian dihitung hasil penjumlahan atau pengurangannya berdasarkan instruksi yang diberikan dan tanda negatif (*flag sign*) dari kedua operand. Pada perkalian, hasil eksponen didapat dari penjumlahan kedua eksponen, dan hasil perhitungan mantissa didapat dari hasil perkalian kedua mantissa. Dan pada pembagian, hasil ekponen didapat dari pengurangan eksponen terbagi oleh eksponen pembagi, dan hasil mantissa didapat dari pembagian mantissa terbagi oleh mantissa pembagi.

Setelah proses perhitungan selesai maka hasil perhitungan akan dinormalisasi, dengan cara menghitung jumlah nol sebelum angka 1 pertama pada mantisa hasil perhitungan, eksponen hasil perhitungan dikurangi jumlah nol sebelum angka 1 pertama, jika hasilnya lebih dari 0 maka hasil eksponen setelah normalisasi adalah sebesar hasil pengurangan tadi, dan mantisa hasil perhitungan digeser sebanyak jumlah 0 sebelum angka 1 pertama pada mantisa hasil perhitungan, akan tetapi jika eksponen hasil pengurangan tersebut kurang dari 0, maka eksponen setelah normalisasi nilainya sama dengan 0 dan hasil mantisa setelah normalisasi digeser sebanyak nilai eksponen. Proses *rounding* (pembulatan) dalam penelitian kali ini adalah pembulatan kearah nol, dengan cara membuang bit yang bernilai terlalu kecil dari kapasitas ketelitian bilangan *floating point* 32 bit sesuai standar IEEE 754, tanpa ada kemungkinan menambahkan nilai 1. Untuk selanjutnya proses *rounding* tidak akan dibahas lebih lanjut atau dianggap tidak ada.

Exception handling berfungsi untuk menangani perhitungan-perhitungan khusus yaitu jika salah satu atau kedua operand adalah bilangan nol, tak hingga, atau NaN (*Not a Number*).

Dalam prosesor-prosesor sederhana yang umumnya tidak memiliki *hardware* khusus untuk menghitung bilangan *floating point* dan hanya mengandalkan manipulasi *software*, satu step pemrosesan seperti yang dijabarkan diatas akan dapat memerlukan satu siklus *clock* atau lebih. Karena itu

perhitungan bilangan *floating point*dalam prosesor-prosesor sederhana membutuhkan waktu yang lama. Dalam perancangan kali ini akan dibuat *hardware* khusus untuk masing masing blok sehingga dapat mengurangi jumlah siklus *clock* yang dibutuhkan dalam setiap perhitungan, dan bahkan FPU yang akan dirancang tidak membutuhkan siklus *clock* untuk menghitung bilangan *floating point*, sehingga kecepatan perhitungan bilangan *floating point* untuk FPU yang dirancang hanya bergantung pada *propragation delay* dari sistem logika yang dibangun.

4.2 Pembangunan Hardware FPU

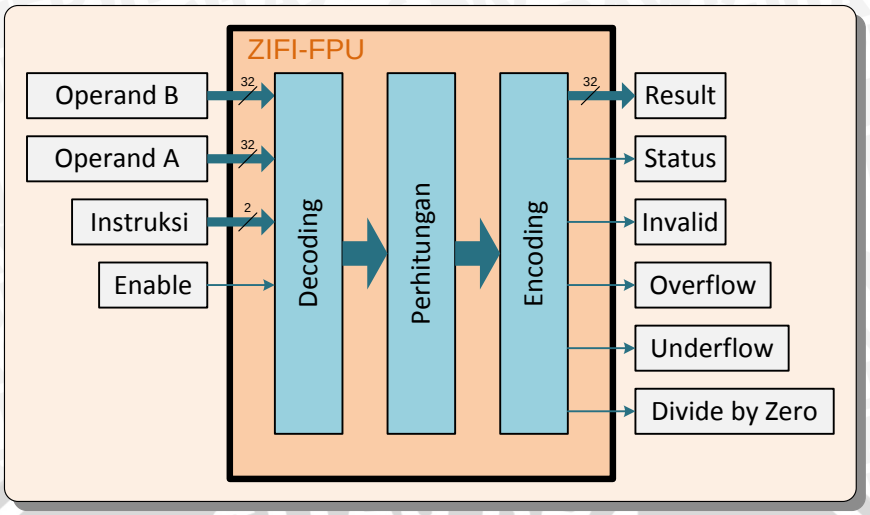
Dalam prosesor-prosesor sederhana semua proses manipulasi logika dilakukan di unit yang biasa disebut ALU, sehingga untuk menghitung bilangan *floating point* yang membutuhkan banyak manipulasi logika akan membutuhkan banyak step. Hal tersebut dapat menjadi lebih rumit dan membutuhkan lebih banyak step lagi bila bus data ALU lebih sedikit dari jumlah bit data yang akan diproses.

Dalam perancangan kali ini setiap jenis manipulasi logika akan ditangani blok *hardware* khusus, sehingga tidak ada satu unit *hardware* yang bertugas lebih dari sekali dalam satu proses perhitungan bilangan *floating point*.

Masing-masing unit *hardware* manipulasi bit bersifat sangat khusus, ia hanya dapat melakukan tugas yang telah ditentukan dalam perancangan, tidak dapat melakukan tugas lain dan bahkan *bus* data-nya benar-benar disesuaikan dengan *bus* data yang akan diproses. Hal ini dilakukan guna mempermudah dalam proses perancangan dan untuk menghemat *slice resources* dalam FPGA yang digunakan.

Dengan demikian maka tugas *control unit* hanya untuk mengendalikan aliran data dari masing-masing blok FPU tanpa perlu untuk mengatur step pemrosesan yang diperlukan ketika terdapat unit *hardware* yang bekerja lebih dari satu kali dalam setiap perhitungan bilangan *floating point*.

Blok diagram dari FPU yang dirancang ditunjukkan dalam gambar 4.3. Dimana FPU yang akan dirancang dapat melakukan empat macam operasi aritmatika: Penjumlahan, pengurangan, perkalian dan pembagian.



Gambar 4.3. Blok Diagram FPU yang akan dirancang.

Unit FPU memiliki beberapa macam masukan antara lain:

1) Dua buah operand dalam standar IEEE 754, dengan bus data 32 bit.
2) Instruksi dengan bus data 2 bit, sebagai penentu operasi aritmatika yang akan dijalankan FPU.
3) *Enable* dengan *bus* data 1 bit, sebagai sinyal *trigger* agar FPU memulai perhitungan aritmatika.
4) *Reset* dengan *bus* data 1 bit, untuk mereset kerja FPU dan mereset hasil keluaran FPU.

Dan FPU ini memiliki model keluaran berupa:

1) Result dengan bus data 32 bit, sebagai tempat keluarnya data hasil perhitungan yang telah dilakukan oleh FPU. Data keluaran ini disajikan dalam format standar IEEE 754.
2) Status dengan bus data 1 bit, sebagai penunjuk apakah FPU telah menyediakan data hasil perhitungan yang valid atau belum.
3) Invalid dengan bus data 1 bit, sebagai petunjuk apakah perhitungan yang dilakukan oleh FPU adalah perhitungan yang tidak valid.
4) Overflow dengan bus data 1 bit, sebagai petunjuk apakah hasil perhitungan yang dilakukan oleh FPU melebihi batas angka maksimal yang dapat disajikan dalam bilangan *floating point* presisi tunggal format IEEE 754.
5) Underflow dengan bus data 1 bit, sebagai petunjuk apakah hasil perhitungan yang dilakukan oleh FPU lebih kecil dari batas angka

minimum yang dapat disajikan dalam bilangan *floating point* presisi tunggal format IEEE 754.

6) Divide by Zero dengan bus data 1 bit, sebagai petunjuk apakah operasi aritmatika yang dilakukan oleh FPU berhubungan dengan operasi pembagian dengan nol atau tidak.

Di dalam FPU tahapan proses perhitungan bilangan *floating point* dapat dibedakan menjadi tiga tahapan, antara lain:

1) Pendekoderan, *operand* masukan diubah kedalam format terpecah antara sign, eksponen, mantissa dan *status flag*. Hal ini agar untuk membuat mudah dan efisien proses di dalam FPU.
2) Penghitungan, terdiri dari perhitungan aritmatika yang terjadi dan proses normalisasi. Penanganan perhitungan khusus juga termasuk dalam tahapan ini.
3) Encoding, Pengemasan hasil keluaran perhitungan atau hasil *exception handling* kedalam format bilangan *floating point* presisi tunggal standar IEEE 754.

4.2 Algoritma FPU

FPU yang akan dirancang dapat menghitung 4 jenis perhitungan dasar, yaitu penjumlahan, pengurangan, perkalian dan pembagian. Perhitungan pengurangan dan penjumlahan dapat

FPU yang akan dirancang haruslah memiliki model kerja tertentu. FPU yang akan dirancang pada penelitian ini memiliki blok-blok pemrosesan yang bekerja berdasarkan *control unit*. Control unit akan menerjemahkan instruksi yang telah diberikan melalui *port Instruction* dan mengendalikan aliran data dari *operand* untuk kemudian diproses melalui blok-blok FPU.

Pada perhitungan penjumlahan terdapat kemungkinan operasi perhitungannya berubah menjadi pengurangan, sebaliknya pada operasi pengurangan juga terdapat kemungkinan operasi perhitungannya berubah menjadi penjumlahan. Hal ini bergantung dari operasi perhitunganya dan kedua keadaan *flag sign* operand masukan. Sehingga dalam penjabarannya kedua operasi ini akan digabungkan dalam satu penjelasan.

Dan untuk perhitungan-perhitungan yang memerlukan penanganan khusus memiliki unit tersendiri yang akan bekerja berdasarkan *status flag* kedua operand.

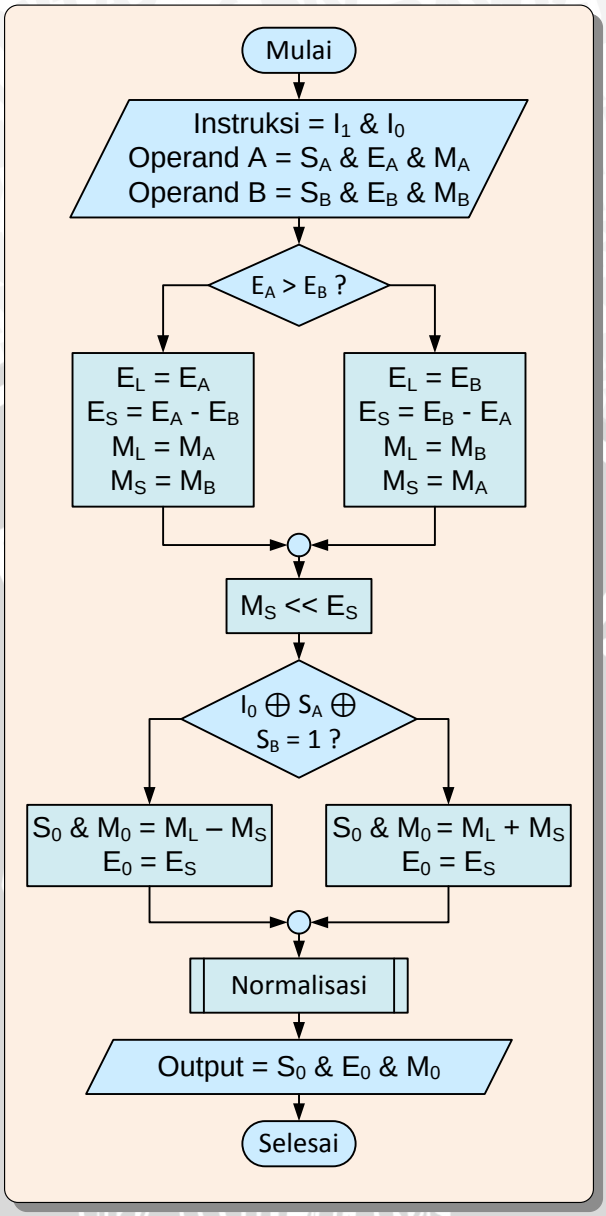
Dengan demikian maka FPU yang dirancang memiliki empat penjabaran untuk algoritma perhitungannya. Yaitu algoritma penjumlahan dan pengurangan, algoritma perkalian, algoritma pembagian, dan algoritma *exception handler*.

4.2.1 Algoritma Penjumlahan dan Pengurangan

Algoritma konvensional penjumlahan dan pengurangan dua buah bilangan *floating point* terdiri dari empat tahap yaitu Perbandingan eksponen, penjajaran mantisa, penjumlahan atau pengurangan mantisa, normalisasi. Mengingat bahwa bilangan *floating point* terdiri dari *sign,* eksponen dan mantisa, tahapan komputasi untuk perhitungan penjumlahan dan pengurangan dua buah bilangan *floating point* adalah sebagai berikut:

1) Pembandingan, bandingkan kedua eksponen, set eksponent terbesar sebagai eksponen hasil tentatif.
2) Penyelarasan, geser mantisa yang memiliki eksponen terkecil kearah kanan sebesar selisih kedua eksponen.
3) Perhitungan, tambahkan atau kurangkan kedua mantisa berdasarkan instruksi yang diberikan dan *flag sign* kedua mantisa untuk mendapatkan mantisa hasil tentatif.
4) Normalisasi. Hitung angka 0 yang mendahului angka 1 pertama dalam mantissa hasil tentatif. Jika hasilnya lebih kecil dari eksponen tentatif maka geser mantisa hasil tentatif sebanyak angka 0 yang mendahului angka 1 dan ekponen hasil adalah ekponen tentatif dikurangi angka 0 yang mendahului angka 1, Jika hasilnya lebih besar dari eksponen tentatif maka geser mantisa tentatif ke kiri sebanyak nilai ekponen tentatif, dan eksponen hasil adalah `00h` (eksponen terbias).

Algoritma yang digunakan untuk menjumlah dan mengurangi dua buah bilangan *floating point* bila dijabarkan dalam bentuk diagram alir ditunjukkan pada Gambar 4.4.



Gambar 4.4. Diagram alir algoritma penjumlahan dan pengurangan

Tahapan penyelarasan dan normalisasi membutuhkan shifter yang besar. Dalam tahap peyelarasan mantisa membutuhkan *shifter* kanan yang memiliki bus data dua kali lebih besar dari jumlah bit mantisa, yaitu sebesar 48 bit. Hal ini dikarenakan bit-bit yang bergeser keluar harus dipelihara nilainya agar tidak terjadi kesalahan dalam penjumlahan dan pengurangan yang akan dilakukan. Karena hal ini *arithmetic unit* yang bertugas untuk penjumlahan dan pengurangan harus memiliki bus data sebesar 48 ditambah 1 bit untuk mendeteksi *carry* atau *borrow*, atau sebesar 49

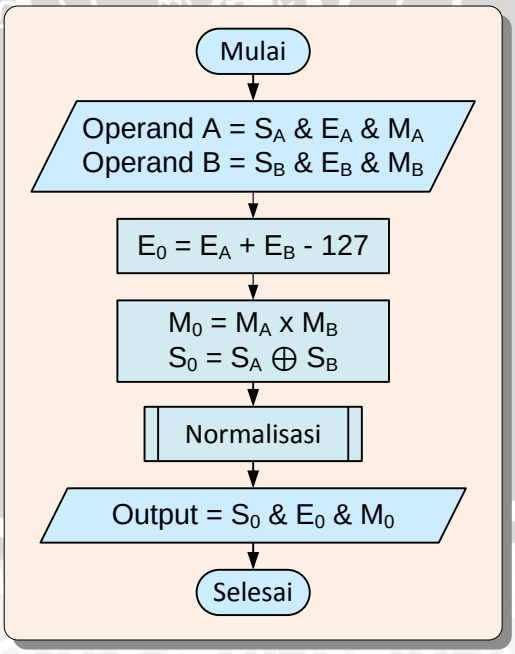
bit.Dan shifter kiri yang diperlukan untuk proses normalisasi membutuhkan bus data masukan sebesar 49 bit.

4.2.5 Algoritma Perkalian

Tahapan yang harus dilakukan dalam perhitungan perkalian dua buah bilangan *floating point* relatif lebih sederhana dibandingkan dengan tahapan dalam perhitungan penjumlahan dan pengurangan. Algoritma untuk melakukan perhitungan perkalian dua buah bilangan *floating point* adalah sebagai berikut:

1) Tambahkan kedua eksponen dan mengurangi 127 (bias).
2) Mengalikan kedua mantisa dan menentukan tanda hasil (*sign out*).
3) Normalisasi hasil perhitungan. Proses ini sama persis dengan proses normalisasi pada algoritma perhitungan penjumlahan dan pengurangan.

Algoritma yang digunakan untuk mengalikan dua buah bilangan *floating point* bila dijabarkan dalam bentuk diagram alir ditunjukkan dalam Gambar 4.5.



Gambar 4.5 Diagram alir untuk algoritma perhitungan perkalian.

Hardware untuk perkalian yang digunakan dalam perancangan ini adalah *hardware* perkalian khusus yang sudah disediakan oleh FPGA yang

digunakan. *Hardware* perkalian ini bersifat paralel dan tidak memerlukan *clock* untuk dapat bekerja. Ciri *hardware* perkalian yang digunakan ini adalah memiliki dua bus data masukan sebesar 24 bit dan satu bus data keluaran sebesar 48 bit.

4.2.7 Algoritma Pembagian

Algoritma konvensional untuk perhitungan pembagian dua buah bilangan *floating point*persis dengan algoritma perkalian kecuali untuk perhitungan kedua mantisa yang dalam hal ini adalah pembagian. Namun karena FPGA yang digunakan dalam penelitian ini tidak mempunyai *hardware* pembagian maka diperlukan pembangunan *hardware* khusus untuk pembagian. Algoritma *hardware* pembagian yang diterapkan dalam penelitian ini menggunakan algoritma yang dipakai dalam kebanyakan sekolah, yang merupakan pembagian melalui beberapa pengurangan atau biasa dikenal dengan sebutan *porogapit*. *Hardware* pembagian diimplementasikan sedemikian rupa agar bersifat paralel atau tidak membutuhkan *clock* untuk bekerja, dengan konsekuensi pemakaian *resource slice* FPGA yang lebih besar.

Algoritma untuk melakukan pembagian dua buah bilangan *floating point* (Operand A / Operand B) adalah sebagai berikut:

1) Hitung jumlah 0 yang mendahului angka 1 pertama di kedua mantisa masukan.
2) Geser kekiri bit mantisa kedua masukan sesuai dengan jumlah 0 yang mendahului angka 1 pertama.
3) Hitung hasil pembagian kedua mantisa, dan tentukan tanda hasil perhitungan.
4) Hitung eksponen hasil dengan persamaan:

   $$E_0 = E_a - E_b + bias - Z_a + Z_b$$

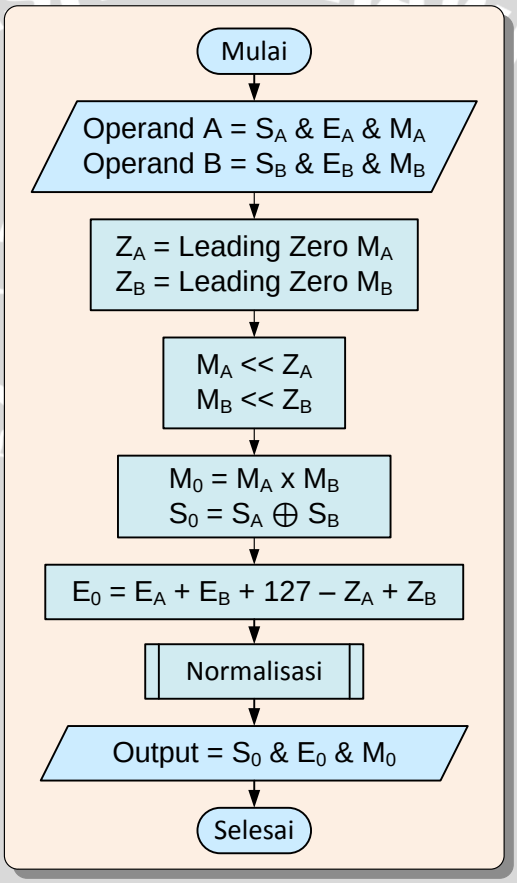
   Dengan:

   - E_0 sebagai eksponen hasil.
   - E_a adalah eksponen dari operand A.
   - E_b adalah eksponen dari operand B.
   - $bias$ adalah konstanta yang bernilai 127.

- Z_a adalah jumlah 0 yang mendahuli angka 1 pertama pada mantisa A.
- Z_B adalah jumlah 0 yang mendahuli angka 1 pertama pada mantisa B.

5) Normalisasi hasil perhitungan. Proses ini sama persis dengan proses normalisasi pada algoritma perhitungan penjumlahan dan pengurangan.

Algoritma yang digunakan untuk melakukan pembagian dua buah bilangan *floating point* bila dijabarkan dalam bentuk diagram alir ditunjukkan dalam Gambar 4.6.



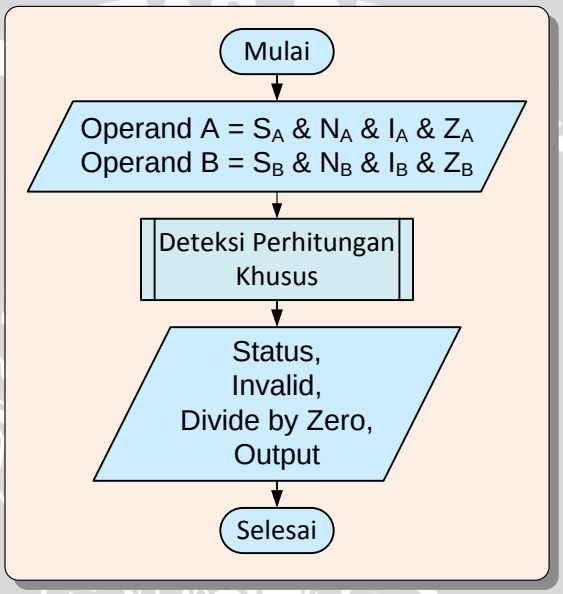
Gambar 4.6. Diagram alir untuk algoritma perhitungan pembagian

4.2.7 Algoritma *Exception Handler*

Algoritma ini berfungsi untuk menghitung hasil perhitungan yang tidak dapat ditangani oleh algoritma konvensional untuk masing-masing perhitungan. *Exception handler* bekerja berdasarkan sinyal *status flag* yang terdiri dari *flag sign, flag Not a Number, flag Infinity,* dan *flag zero.*

Keluaran dari *exception handler* terdiri dari sinyal status sebagai penanda apakah *exception handler* melakukan perhitungan khusus atau tidak, sinyal invalid sebagai penanda bahwa perhitungan yang dilakukan adalah perhitungan yang tidak valid, sinyal divide by zero sebagai penanda bahwa perhitungan yang dilakukan adalah perhitungan pembagian dengan angka 0, dan output sebagai keluaran dari *exception handler* yang sudah terformat dalam standar IEEE 754 untuk *single precision*.

Secara singkat dan sederhana, penjabaran algoritma *exception handler* dalam bentuk diagram alir ditunjukkan dalam Gambar 4.7.



Gambar 4.7. Diagram alir algoritma *exceptionhandler*.

Penjabaran mendetil sifat-sifat *exception handler* akan dijelaskan pada bagian perancangan komponen FPU. Tujuan disendirikannya algoritma *exception handler* ini adalah untuk mempermudah dalam perancangan sistem FPU.

4.3 Perancangan Komponen FPU

Untuk memenuhi algoritma yang telah dijelaskan diatas, maka FPU yang dirancang akan membutuhkan 8 unit kerjayang memilki fungsi dan tugas berbeda-beda. Unit-unit tersebut antara lain adalah *Operand Decoder, Logical Right Shifter, Arithmetic Unit, Logical Left Shifter, Leading Zero Counter, Exception Handling,Floating Point Encoder* dan Unit Input Output.

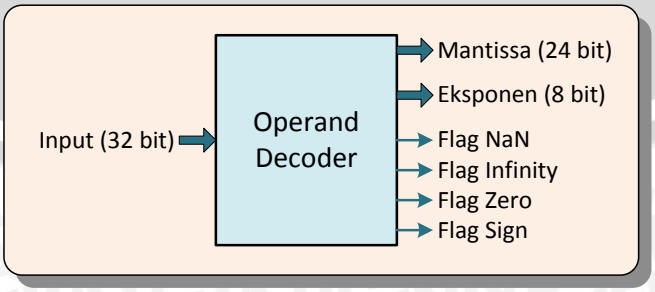
Seluruh unit tersebut akan berkerja sama satu dengan yang lain dalam mengeksekusi perintah dari pengguna, sehingga FPU ini dapat bekerja sesuai dengan yang diinginkan.

4.3.1 Unit *Operand Decoder*

Operand Decoder adalah unit yang berfungsi memecah masukan operand masukan yang memenuhi standard IEEE 754, menjadi pecahan informasi yang berupa mantissa, eksponen, dan *flag* yang terdiri dari *flag sign* (tanda negatif), *flag zero* (tanda nilai nol), *flag infinity* (tanda nilai tak hingga), dan *flag Not a Number* (tanda nilai bukan sebuah nomor).

Unit *operand decoder* dalam sistem FPU yang dirancang terdapat dua, satu untuk memecah operand dari masukan operand A dan yang lain untuk memecah operand dari masukan operan B. Penggunaan dua unit *operand decoder* ini adalah agar FPU dapat melakukan pemecahan informasi kedua masukan operand secara serentak atau paralel. *Unit operand decoder* ini akan bekerja secara langsung, yaitu akan memproses masukan *operand* tanpa menunggu perintah dari *Control unit*, sedangkan keluaran dari *operand decoder* akan dikoneksikan kepada *unit-unit* lain pada FPU sesuai dengan perintah *control unit.*

Bus masukan masing-masing*operand decoder* adalah 32 bit, sesuai dengan besar kapasitas *operand* masukan. Dan keluaran *operand decoder* antara lain, mantissa dengan *bus data* 24 bit, eksponen dengan bus data 8 bit, dan flag yang terdiri dari, *flag Not a Number, Flag Sign, Flag Infinity, Flag Zero*, yang besarnya masing-masing 1 bit. Struktur sistem keseluruhan operand decoder in ditunjukkan dalam gambar 4.8.



Gambar 4.8. Struktur sistem operand decoder

Ketika nilai input bit ke-30 sampai bit ke-24 adalah `00h` menunjukkan nilai eksponen sama dengan -126 bukan -127, karena itu keluaran eksponen ketika kondisi ini adalah `01h` yang menunjukkan eksponen bernilai -126.

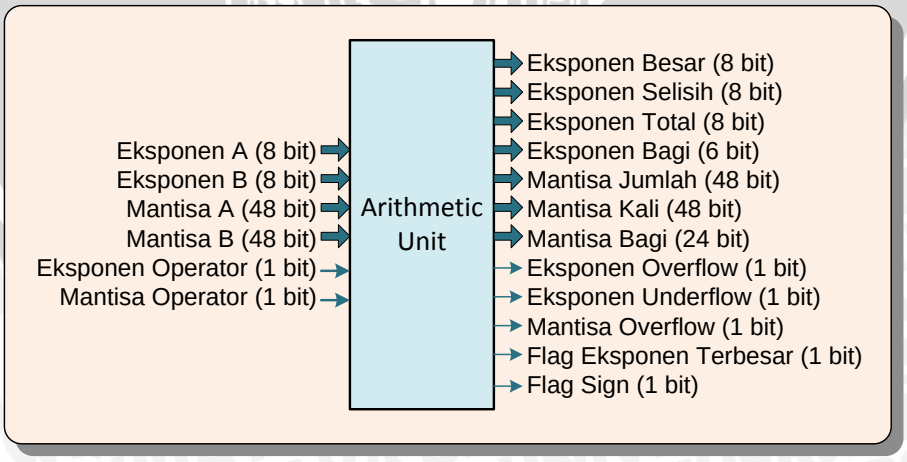
4.3.2 *Arithmetic Unit*

Arithmetic Unit berfungsi untuk menangani segala perhitungan dasar yang diperlukan dalam kerja FPU. Perhitungan-perhitungan dasar ini antara lain, perbandingan, penjumlahan dan pengurangan eksponen, dan penjumlahan, pengurangan, perkalian, serta pembagian mantissa.

Penjumlahan eksponen berfungsi ketika FPU melakukan proses perkalian. Pengurangan eksponen berfungsi ketika FPU melakukan proses pembagian dan untuk menentukan bilangan yang lebih besar untuk kemudian disamakan ekponennya dengan bilangan yang lebih kecil dengan melakukan penyesuaian terhadap mantissa.

Masukan *Arithmetic Unit* ini ditentukan oleh *control unit*, tidak serta merta dari unit *operand decoder*, sebab misalnya dalam perhitungan penjumlahan dan pengurangan, salah satu mantissa harus digeser sebanyak selisih kedua eksponen sebelum kedua mantissa tersebut dijumlahkan atau dikurangkan oleh *arithmetic unit*.

Masukan ALU ini ditentukan oleh unit control. Dan data keluaran dari ALU ini juga akan ditangani oleh unit control. Struktur sistem keseluruhan operand decoder ini ditunjukkan dalam Gambar 4.9.



Gambar 4.9. Struktur sistem ALU

Eksponen besar nilainya sama dengan nilai terbesar antara masukan eksponen A dengan eksponen B. Eksponen nilainya adalah sama dengan

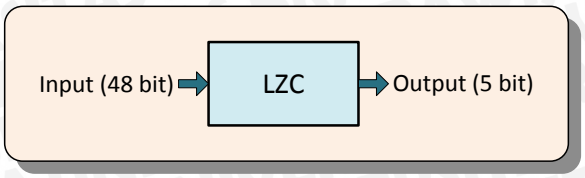
selisih antara masukan ekponen A dan eksponen B. Eksponen total menunjukkan hasil penjumlahan atau pengurangan dua eksponen masukan, penjumlahan atau pengurangan ini tergantung dari masukan eksponen operator. Eksponen bagi adalah nilai eksponen dari hasil pergeseran ke kiri dari dua mantisa sebelum pembagian. Mantisa jumlah adalah hasil penjumlahan atau pengurangan mantisa A dengan mantisa B, penjumlahan atau pengurangan yang dilakukan tergantung pada masukan mantisa operator. Eksponen overflow menunjukkan *carry* dari hasil penjumlahan kedua eksponen masukan. Eksponen underflow menunjukkan *borrow* dari hasil pengurangan eksponen A terhadap eksponen B. Mantisa overflow menunjukkan *carry* dari hasil penjumlahan kedua mantisa masukan. *Flag* eksponen terbesar menunjukkan eksponen mana diantara eksponen A dan eksponen B yang bernilai lebih besar. Dan *flag sign* menunjukkan tanda hasil dari perhitungan penjumlahan dan pengurangan dua masukan mantisa.

4.3.3 Unit *Leading Zero Counter* (LZC)

Unit LZC berfungsi untuk menghitung jumlah nol yang mendahului sebelum angka 1 pertama dalam bit mantissa hasil perhitungan. Misalnya data masukan LZC adalah`0000 0010 1101 0111 1000 0000b` maka keluaran LZC adalah `00110b` atau bernilai 6 dalam sistem bilangan desimal. Artinya bahwa masukan dari LZC memiliki 6 deret angka nol sebelum angka 1 pertama.

Unit LZC bekerja ketika FPU melakukan normalisasi hasil perhitungan. Yaitu untuk mengetahui banyaknya pergeseran kekiri yang harus dilakukan oleh mantissa hasil perhitungan dan besar pengurangan eksponen hasil perhitungan setelah mantissa digeser.

Jadi masukan LZC ini adalah mantissa hasil perhitungan *Arithmetic Unit*, dan keluarannya akan menjadi masukan *Logical left shifter*. Unit LZC memiliki bus masukan sebesar 48 bit dan bus keluaran sebesar 5 bit. Secara keseluruhan, struktur sistem LZC ditunjukkan dalam gambar 4.10.

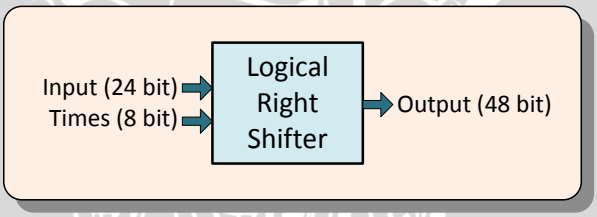


Gambar 4.10 Struktur sistem unit LZC

4.3.4 Unit *Logical Right Shifter*

Unit ini berfungsi untuk menggeser bit mantissa kekanan atau kearah bit yang bernilai lebih kecil. Penggeseranbit ini dilakukan ketika penyesuaian mantissa yang memiliki eksponen lebih kecil pada saat perhitungan penjumlahan dan pengurangan. Mantissa digeser sebanyak selisih eksponen A dan eksponen B.Pergeseran bit ke kanan ini diperlukan ketika FPU melakukan proses perhitungan penjumlahan atau pengurangan.

Oleh karena diharapkan bit-bit yang tergeser ke kanan tidak boleh hilang untuk menghindari kesalahan perhitungan, maka bus keluaran unit *Logical right shifter* butuh dua kali lebih besar dari bus masukan. Jadi dengan bus masukan 24 bit maka bus keluaran unit *logical right shifter* adalah 48 bit. Secara keseluruhan struktur sistem unit *logical right shifter* ditunjukkan dalam Gambar 4.11.



Gambar 4.11. Struktur sistem *logical right shifter*

Input menunjukkan masukan data yang akan digeser ke arah kanan oleh *logical right shifter*. Masukan *Times* menunjukkan banyaknya pergeseran yang harus dilakukan oleh *logical right shifter*. Dan Output adalah hasil pergeseran yang telah dilakukan.

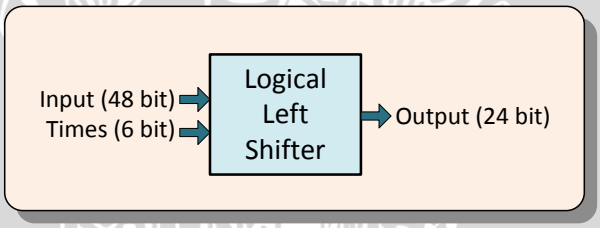
4.3.5 Unit *Logical Left Shifter*

Unit *logical left shifter* berfungsi untuk menggeser bit mantissa kekiri atau kearah bit ang bernilai lebih besar dari hasil perhitungan dari unit *Arithmetic Unit*. Penggeseran bit ini dilakukan untuk keperluan proses

normalisasi hasil perhitungan. Mantissa hasil perhitungan akan digeser kekiri sebanyak jumlah nol yang mendahului nilai 1 pertama dalam mantissa hasil perhitungan tersebut atau sebanyak nilai eksponen hasil perhitungan, hal ini tergantung apakah nilai eksponen hasil perhitungan lebih besar atau lebih kecil dari jumlah nol yang mendahului angka 1 pertama dalam mantissa hasil perhitungan.

Bus keluaran Unit *Logical left shifter* ini sebesar 24 bit sesuai dengan jumlah bit mantissa pada bilangan *floating point* standar IEEE 754. Sedangkan bus masukan unit *logical left shifter* ini adalah 48 bit, maka keluaran unit *logical left shifter* ini adalah 24 bit pertama terhitung dari kanan setelah masukan unit *logical left shifter* digeser, dan 24 bit LSB akan diabaikan nilainya.

Jadi masukan unit *logical left shifter* ini memiliki bus data 48 bit dan memiliki masukan untuk penentu banyaknya pergeseran dengan bus data sebesar 8 bit. Secara keseluruhan struktur sistem unit *logical left shifter* ditunjukkan dalam Gambar 4.12.



Gambar 4.12. Struktur sistem *logical left shifter*.

Input menunjukkan masukan data yang akan digeser ke arah kiri oleh *logical left shifter*. Masukan Times menunjukkan besar pergeseran yang harus dilakukan oleh *logical right shifter*. Dan *Output* adalah hasil keluaran data setelah mengalami pergeseran dan pemotongan sebesar 24 bit.

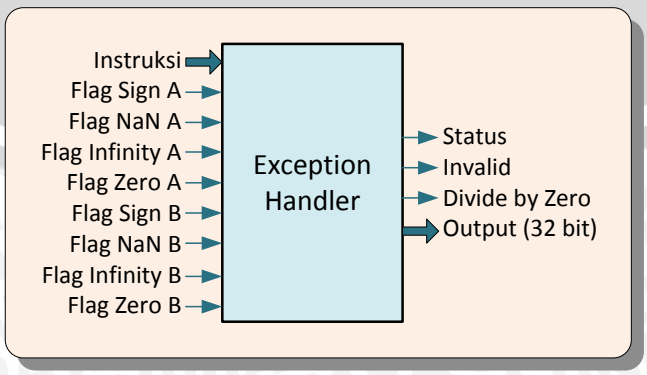
4.3.6 Unit Exception Handling

Unit ini berfungsi untuk mengatasi perhitungan-perhitungan khusus yang tidak bisa diselesaikan dengan perhitungan normal. Perhitungan-perhitungan khusus tersebut antara lain:

1) Salah satu masukan adalah *Not a Number*, maka keluaran perhitungan adalah *Not a Number*.

2) Nol dikali dengan *Infinity*, Maka keluaran perhitungan adalah *Not a Number.*
3) Nol dibagi dengan Nol, maka keluaran perhitungan adalah *Not a Number*.
4) Bukan Nol dibagi dengan Nol, maka keluarannya adalah *Infinity* dan keluaran *Sign*akan bernilai 1 (negatif) jika *sign* salah satu atau kedua masukan bernilai 1 (negatif).
5) *Infinity* dibagi dengan *Infinity*, maka keluaran perhitungan adalah *Not a Number.*
6) Bukan *Infinity* dibagi dengan *Infinity*, maka keluaran perhitungan adalah Nol dan keluaran *Sign*akan bernilai 1 (negatif) jika *Sign* salah satu atau kedua masukan bernilai 1 (negatif).
7) Positif *Infinity* ditambah negatif *Infinity*, maka keluaran perhitungan adalah *Not A Number*.
8) Positif *Infinity* dikurangi positif *Infinity*, atau negatif *Infinity* dikurangi negatif *Infinity*, maka keluaran perhitungan adalah *Not A Number*.
9) Positif *Infinity* ditambah positif *Infinity* atau positif *Infinity* dikurangi negatif *Infinity*, maka keluaran perhitungan adalah positif *Infinity*.
10) Negatif *Infinity* dikurangi positif *Infinity* atau negatif *Infinity* ditambah negatif *Infinity*, maka keluaran hasil perhitungan adalah negatif *Infinity*.

Jadi *exception handler* memerlukan masukan berupa *flag* dari kedua masukan operand yang didapat dari *operand decoder* dan akan member sinyal keluaran yang akan dapat mempengaruhi keluaran unit *floating point encoder*. Secara keseluruhan struktur sistem exception decoder ditunjukkan dalam Gambar 4.13.

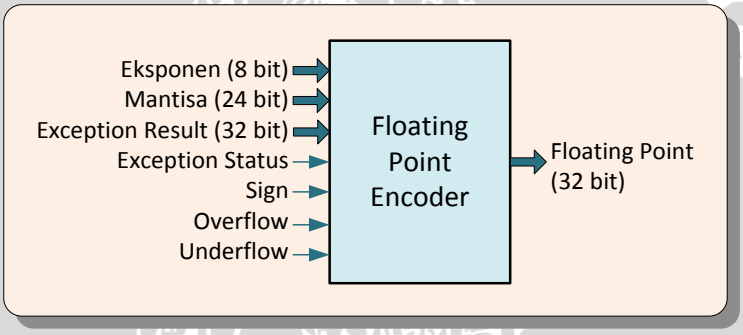


Gambar 4.13. Struktur sistem *exception handler*

4.3.7 Unit *Floating Point Encoder*

Unit *Floating point encoder* adalah unit yang berfungsi untuk meng-enkoderkan hasil perhitungan FPU atau hasil dari *exception handler* dalam bentuk standarIEEE 754 untuk bilangan *floating point* presisi tunggal. Secara normal *floating point encoder* akan mengemas sinyal *sign,* eksponen, dan mantissa hasil perhitungan kedalam bentuk *floating point standard* IEEE 754, kecuali jika unit *exception handler* memberikan sinyal peringatan bahwa salah satu atau kedua operand masukan bernilai khusus (*Not a Number, Zero,* atau *Infinity*) maka *floating point encoder* akan menghasilkan keluaran sesuai yang diperintahkan oleh unit *exception handler*.

Secara keseluruhan struktur sistem *floating point encoder* ditunjukkan dalam Gambar 4.14.



Gambar 4.14. Struktur sistem *floating point encoder.*

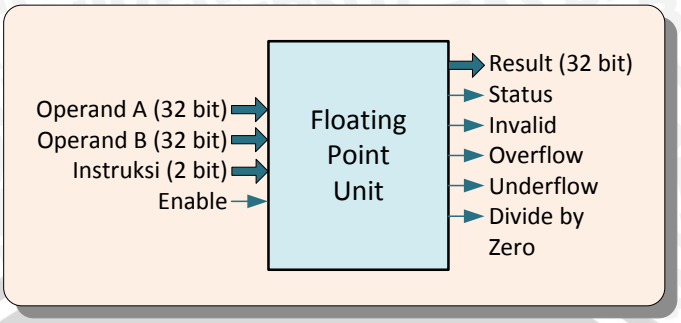
4.3.7 *Control Unit*

Control Unit adalah unit yang berfungsi sebagai pengatur utama jalannya FPU. Control unit akan mengatur unit-unit lainya berdasarkan instruksi masukan yang diberikan.

Kinerja dari control unit ini merepresentasikan kinerja dari FPU. Karena itu pengujian dari control unit adalah sama dengan pengujian keseluruhan.

Control unit adalah unit yang berfungsi sebagai control utama jalannya FPU. Control unit akan mengatur unit-unit lainnya berdasarkan instruksi yang didapatkan dan akan mengatur kerja unit-unit lain dan aliran data unit satu dengan unit lain.

Jadi struktur sistem *control unit* atau struktur sistem FPU secara keseluruhan ditunjukkan dalam Gambar 4.15.



Gambar 4.15. Struktur sistem *control unit.*

Kerja *control unit* secara singkat adalah sebagai berikut:

1. Menjadikan data Operand A sebagai masukan unit *operand decoder A.* dan menjadikan data Operand B sebagai masukan unit *operand decoder B*.
2. Menjadikan keluaran *flag* (*Sign, Not a Number, Infinity,* dan *Zero*) dari unit *operand decoder A* dan unit *operand decoder B* sebagai masukan unit *exception handler*.
3. Menjadikan keluaran *output* dan *status* dari unit *exception handler* sebagai masukan *exception result* dan *exception status* dari unit *floating point encoder*.
4. Keluaran eksponen A dan eksponen B dari unit *operand decoder* menjadi masukan eksponen A dan eksponen B *arithmetic unit*.
5. Ketika menerima instruksi penjumlahan dan pengurangan *control unit* mengatur:
    a. Eksponen selisih dari *arithmetic unit* menjadi masukan *Times* dari unit *logical right shifter*.
    b. Menentukan eksponen terbesar dan terkecil berdasarkan keluaran *Flag* eksponen terbesar dari *aritmetic unit*.
    c. Mantisa dengan eksponen terkecil menjadi *Input* dari unit *logical right shifter*.
    d. Mantisa dengan eksponen terkecil dan *output* dari unit *logical right shifter* menjadi masukan mantisa A dan mantisa B dari *arithmetic unit*.

e. Keluaran mantisa jumlah dari *arithmetic unit* menjadi masukan unit *leading zero counter* dan unit *logical left shifter*.
f. Yang bernilai lebih kecil dari keluaran unit *leading zero counter* dan eksponen terbesar dari *arithmetic unit,* menjadi masukan *Times* dari unit *logical left shifter*.
g. Keluaran tanda hasil perhitungan (*Flag Sign*) dari *arithmetic unit* menjadi masukan *sign* dari unit *floating point encoder*.
h. Keluaran unit *logical left shifter* menjadi masukan mantisa dari unit *floating point encoder*.
i. Mengurangkan eksponen terbesar dari *arithmetic unit* dengan keluaran *leading zero counter* dan menjadikannya sebagai masukan eksponen dari unit *floating point encoder*.

6. Ketika menerima instruksi perkalian *control unit* mengatur:
   a. Keluaran mantisa A dan mantisa B dari unit *operand decoder* menjadi masukan mantisa A dan mantisa B dari *arithmetic unit*.
   b. Keluaran mantisa kali dari *arithmetic unit* menjadi masukan unit *leading zero counter* dan unit *logical left shifter*.
   c. Yang bernilai lebih kecil dari keluaran unit *leading zero counter* dan keluaran eksponen total dari *arithmetic unit*, menjadi masukan *times* dari unit *logical left shifter*.
   d. Keluaran unit *logical left shifter* menjadi masukan mantisa dari unit *floating point encoder*.
   e. Mengurangkan keluaran eksponen total dari *arithmetic unit* dengan keluaran unit *leading zero counter* dan menjadikannya sebagai masukan eksponen dari unit *floating point encoder*.
7. Ketika menerima instruksi pembagian *control unit* mengatur:
   a. Keluaran mantisa A dan mantisa B dari unti *operand decoder* menjadi masukan mantisa A dan mantisa B dari *arithmetic unit*.
   b. Menentukan masukan *times* dari unit *logical left shifter* berdasarkan keluaran eksponen total dan eksponen bagi dari *arithmetic unit* dan keluaran unit *leading zero counter*.
   c. Keluaran mantisa bagi dari *arithmetic unit* menjadi masukan unit *leading zero counter* dan unit *logical left shifter*.

d. Keluaran unit *logical left shifter* menjadi masukan mantisa dari unit *floating point encoder*.

e. Menentukan masukan eksponen dari unit *floating point encoder* berdasarkan keluaran eksponen total dan eksponen bagi dari *arithmetic unit* dan keluaran unit *leading zero counter*.

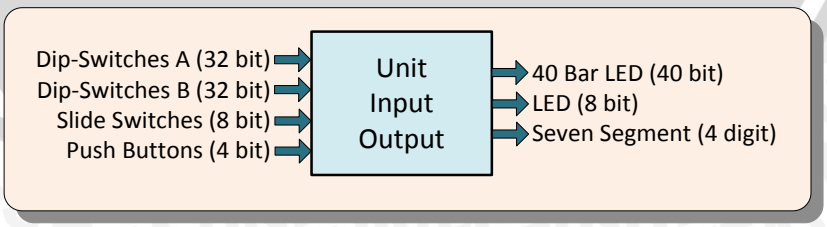
8. Menjadikan keluaran *floating point 32* dari unit *floating point encoder* sebagai keluaran *result* dari FPU.
9. Selain yang telah disebutkan *control unit* juga mengatur hasil dari *sign*, *overflow, underflow*, *divide by zero* dari blok-blok unit yang berhubungan.

4.3.8 Unit *Input-Output*

Unit *input-output* sebenarnya adalah unit diluar FPU yang dirancang, akan tetapi unit ini diperlukan untuk proses pengujian. Unit ini berfungsi untuk memberi masukan FPU mengaktifkan proses FPU dan menampilkan keluaran FPU.

Unit *input-output* ini terdiri dari dua bagian, yang pertama yaitu unit *input-output* yang telah tersedia pada board Digilent Nexys2 yang terdiri dari masukan 8 bit *switches* dan 4 bit *push buttons,* dam keluaran berupa 8 bit LED dan 4 buah seven segment. Yang kedua adalah unit input-output luar yang berguna untuk dapat memberikan keluaran dan masukan dengan bit lebih besar, terdiri dari masukan 32 bit *dip-switches A* dan32 bit *dip-switches B* dan keluaran 40 bit Bar LED.

Secara keseluruhan struktur sistem unit *input-output* ditunjukkan dalam Gambar 4.16.



Gambar 4.16. Struktur sistem unit *input-output*