Ubacivanje više instrukcija po ciklusu u ugrađenu dataflow mašinu i Intel jezgra.

Neposredno uz sam procesor se nalaze dve memorije: data keš i instrukcijski keš. Svaka od ovih keš memorija ima veoma široku magistralu prema procesoru i tipična širina je 192 ili 256 bita u najrasprostranjenijim današnjim procesorima za instrukcijski keš i 512 bita za data keš. Široka reč instrukcijskog keša koja ima veliki broj instrukcija se zatim delimično dekoduje. Cilj delimičnog dekodovanja je da se otkriju granice instrukcija, kako bi se u narednom koraku mogla raditi potpuno paralelno dekodovanje svih instrukcija koje se cele nalaze u širokoj reči instrukcijskog keša. Kasnije će biti razmatrani slučajevi kada se jedna instrukcija nalazi u dve susedne sukcesivne velike reči.

Razmotrimo prvo slučaj bez grananja. Tada se svakog ciklusa uzima nova reč i u jednom ciklusu se određuju granice instrukcija, a u narednom ciklusu pipeline-a radi potpuno dekodovanje i formiranje mikrooperacija. Tako se može formirati niz mikrooperacija po originalnom in-order redosledu izvršavanja, spremnih za izvršavanje. Kada se primeni paralelno preimenovanje, opisano kod tag indexed mašina, nakon ubacivanja instrukcija u ugrađenu dataflow mašinu se one paralelno izvršavaju, poštujući ograničenja trenutnog dinamičkog grafa zavisnosti po podacima. Međutim, granice instrukcija ne moraju da se poklapaju sa granicama reči instrukcijske keš memorije. Zato se na krajevima reči moraju preuzimati delovi instrukcija i stapati, kako bi se dobile instrukcije za dekodovanje.

U slučaju koda sa grananjima, ubacivanje instrukcija u dataflow mašinu je znatno komplikovanije. Za to postoji više razloga:

a. Uslovni skokovi kod kojih je dinamički prediktor grananja predvideo skok, a kraj instrukcije skoka nije kraj široke reči instrukcijskog keša
b. Bio je predviđen skok, a doskače se na početak instrukcije čiji se početak ne poklapa sa početkom široke reči instrukcijskog keša.
c. U reči se nalazi više instrukcija uslovnog skoka
d. U reči se nalaze uskakanje u kod reči na početak instrukcije čiji se početak ne poklapa sa početkom široke reči data keša, a zatim pre kraja reči sledi uslovni skok kod kojeg je dinamički prediktor grananja predvideo skok.

U slučaju a., sve što je preneto u procesor posle predviđenog skoka čitanjem široke reči data keša se mora odbaciti, i ne dekoduje se dalje, jer te instrukcije nisu predviđene za izvršavanje na osnovu spekulacije po kontroli koju radi prediktor grananja. U slučaju da nije predviđen skok, sve se radi kao što je opisano za kod bez grananja.

Slučaj b. je komplementaran slučaju a. Tada se mora odbacivati deo koda koji prethodi tački uskakanja (početku instrukcije posle skoka), jer se uvek čitaju cele široke reči. Kada bi uskakanje bilo na početak reči, nema odbacivanja delova reči, ako nema kasnijih grananja u istoj reči. Tokom kompajliranja se često tačke doskakanja namerno postavljaju na početak široke reči instrukcijskog keša, kako bi se postigla veća prosečna brzina ubacivanja instrukcija iz instrukcijskog keša.

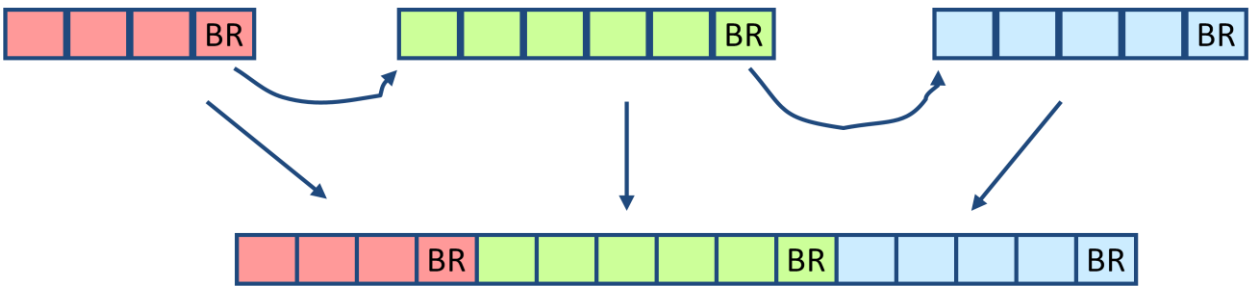
Slučaj c. se direktno izvodi iz slučaja a. Odbacivanje se radi za deo koda iza prvog grananja za koje je prediktor predvideo da će se obaviti skok. U ovom slučaju je potrebno imati prediktor koji pravi više predikcija po ciklusu takta.

Za slučaj d. se praktično sprovodi kombinovanje slučaja b., gde se odbacuje kod pre tačke uskakanja, i slučaja a. ili c., gde se odbacuje deo koda posle predviđenog skoka.

Na osnovu prethodnih stavova, dinamika ubacivanja instrukcija jako zavisi od koda i predviđenog dinamičkog traga od strane prediktora grananja. Usporavanje rada dataflow mašine zbog sporosti ubacivanja instrukcija nije prihvatljivo. Zato se rade sledeće stvari:

1. Dinamički prediktor mora da prednjači i više reči iz data keša mora da se dohvata po ciklusu, ali prema predviđenim ishodima grananja koje daje prediktor.
2. Kako je grubo svaka 6-7 instrukcija predstalja uslovni skok, veoma je verovatno da najnoviji procesori rade dve predikcije po ciklusu i dohvataju dve reči instrukcijskog keša svakog ciklusa.
3. Za programske petlje postoje baferi u kojima se čuva kod petlje, pa kada se posebnom logikom detektuje da je u pitanju petlja, prestane se dohvatanje stalno istog koda iz data keša, sve dok se ne predvidi iskakanje iz petlje. Ovaj bafer je reda veličine oko 1000 instrukcija u današnjim procesorima.
4. Realizuje se trace bafer koji pamti u in order redosledu dekodovane instrukcije u FIFO redosledu na osnovu predikcije grananja. Tako se dolazi do in-order ulaza u ugrađenu dataflow mašinu.

Formiranje trace buffera je pokazano ilustrativno na Sl. 1. Ključni novi element je da se iz data keša radi prepakivanje po predviđenom redosledu izvršavanja u pomoćnoj keš memoriji koja pamti dinamički trag. Za tu memoriju se zatim primenjuje logika opisana na početku ovog odeljka, kao da se radi samo o bazičnim blokovima.



Sl. 1. Trace keš

U Intel familiji procesora postoje dve pomoćne brze memorije između data keša i procesora. Prva se zove predecode fetch buffer i u njoj se određuju granice instrukcija, predikcije za skokove i potrebni delovi koda šalju do trace buffera, koji se kod Intela zove instruction queue. U slučaju Intelove generacije Haswell procesora implementacija dela procesora do ubacivanja instrukcija u ugrađenu dataflow mašinu je prikazan u gornjem delu Sl. 2.

Nakon instrukcijskog queue-a (reda) se CISC instrukcije familije x86 dovode do dekodera koji generišu mikroinstrukcije. Prema statistici korišćenja instrukcija, projektanti procesora su procenili da na prosečno

tri jednostavne instrukcije dolazi jedna kompleksna, koja se konvertuje u do 4 mikroinstrukcije. Mikrokod(ovi) dekodovanih instrukcija se zatim opet smeštaju u novi trace buffer (μop decode queue) potpuno dekodovanih instrukcija. Odatle se po in order redosledu ubacuju u ROB i rade se preimenovanja (nije nacrtano na Slici 2.), a u slučaju Load, Store i Branch instrukcija ubacuju u odgovarajuće bafere. Uloga Load i Store bafera je opisana u poglavlju o zavisnostima po podacima preko memorije, a Branch buffer pre svega čuva predikcije, kako bi se izračunati flagovi poredili sa predviđenim.

U ROB se ubacuju po 4 mikrooperacije svakog ciklusa, ako ima bar 4 prazne lokacije. Istovremeno se mikrooperacije (najčešće ekvivalent instrukcije) šalju do Schedulera, koji je ustvari Issue queue. U fazi preimenovanja, svaka nova instrukcija ubačena u ugrađenu dataflow mašinu dobija svoj novi fizički registar za rezultat, a skup fizičkih registara na Sl. 2. je nazvan Integer registers. Naravno, ako su u pitanju vektorske operacije, podaci se šalju direktno u SWAR vektorske registre i rezultati vektorskih instrukcija vraćaju u data keš.

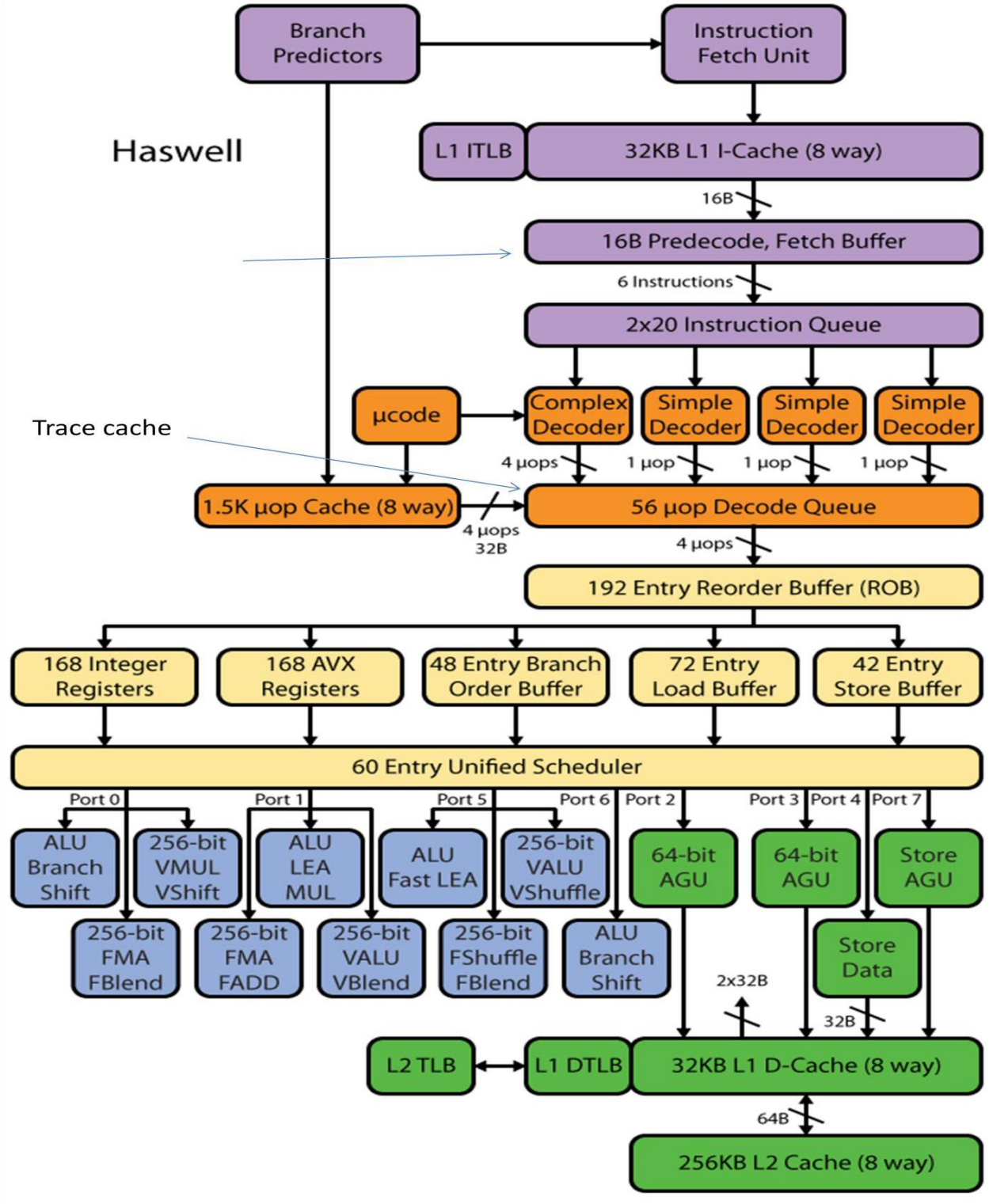
Izvršne jedinice su organizovane kao nezavisne, ili su priključene grupama jedinica. Grupe jedinica su organizovane tako da se retko javlja potreba da se dve ili više jedinica istovremeno koriste, pa tako mogu da dele isti port. Deljenje portova je potrebno da bi se smanjila kompleksnost multiportne registarske memorije, odnosno interkonekcione mreže koja povezuje izlaze fizičkih registara sa odgovarajućim ulaznim portovima izvršnih jedinica i za prosleđivanje rezultata do fizičkih registara, kada nisu u pitanju vektorske operacije.

Uočljivo je postojanje vektorskih instrukcija kod Haswell procesora za 128 bita i 256 bita, a najnovija generacija ima još i instrukcije za 512 bita. U cilju smanjivanja opterećenja multiportne memorije, odnosno interkonekcione mreže, ubačene su jedinice FMA, množači/sabirači koji rade sumu proizvoda, bez pamćenja proizvoda u fizičkim registrima, jer nema drugih instrukcija zavisnih po podacima od pojedinačnih proizvoda. Te operacije su jako korisne kod skalarnih proizvoda vektora, matričnih množenja i sl.

Portovi 2, 3 i 7 se koriste za adresne generatore za load i store, koji su integerske jedinice. Kako izračunavanje pointera u ovim jedinicama predstavlja generisanje paterna adresa po kome će se čitati ili upisivati podaci u memorijskim operacijama, ne može da postoji zavisnost po podacima između samih podataka i njihovih adresa. Zato je scheduler za izračunavanje adresa u osnovi nezavisan po podacima od schedulera za podatke. Pored toga port 4 je namenjen za podatke koji se šalju u writeback store buffer (nije prikazan na slici).

Preostala 4 porta sadrže grupe jedinica i vezani su za podatke, bilo da su u pitanju skalarni ili vektorski podaci. Zanimljivo je da u jednom ciklusu može da se radi paralelno razrešavanje da li je tačna predikcija za dva uslovna skoka po ciklusu, što verovatno znači da prediktor radi dve predikcije po ciklusu. Za procesore proizvedene u poslednjih 7-8 godina se način realizacije predikcije grananja krije kao najveća poslovna tajna. Takođe se kriju svi načini korišćenja magistrale koja je u Haswell procesoru 256 bita, a u današnjim procesorima 512 bita. Verovatno je da se magistrala dinamički deli na delove koji ostvaruju komunikaciju sa izvršnim jedinicama. Osim toga, scheduler verovatno planira izvršavanje mikrooperacija tako da se isti podatak emituje ka više izvršnih jedinica kada postane data ready, slično višestrukim

common data busevima kod Tomasula, ali dinamički promenljive širine.



Sl. 2. Haswell jezgro Intel procesora, ključne komponente

Kompleksnost nekoliko komponenti superskalarnog jezgra raste sa N^2, gde je N broj instrukcija koje su istovremeno kandidati za raspoređivanje u Unified Scheduleru (issue queue) ili pak broj instrukcija koje se po ciklusu ubacuje u paketu u procesor. To je osnovni razlog zašto se u današnjim procesorima stavlja više superskalarnih procesora (jezgara), jer je porast kompleksnosti u tom slučaju približno linearno zavisna od broja jezgara. Balansiranje opterećenja jezgara u višenitnim aplikacijama nije lako rešiv problem, ali to nije predmet ove knjige.